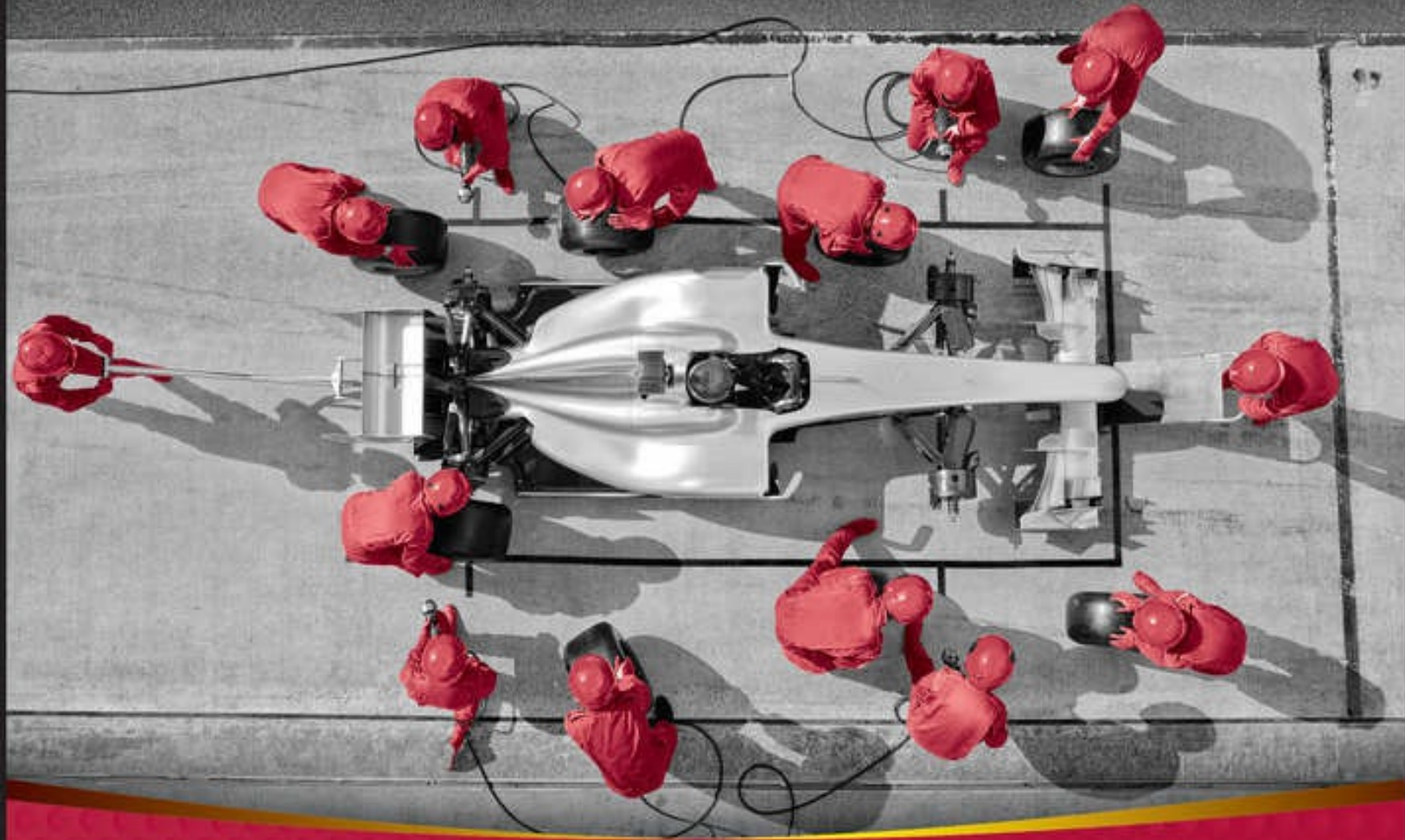


Join the discussion @ p2p.wrox.com



Wrox Programmer to Programmer™



Professional

DNN7

Open Source .NET CMS Platform

Shaun Walker, Bruce Chapman, Cathal Connolly, Peter Donker, Israel Martinez, Charles Nurse, Chris Paterra, Clinton Bland, Ashish Prasad, Nathan Rover, Mitchel Sellers, Dennis Shiao, Will Strohl, Erik van Ballegoij, Scott Willhite, Ralph Williams, Jr.

Table of Contents

[Introduction](#)

[Who This Book Is For](#)

[What This Book Covers](#)

[How This Book Is Structured](#)

[What You Need to Use This Book](#)

[Conventions](#)

[Source Code](#)

[Errata](#)

[p2p.wrox.com](#)

[Chapter 1: An Inside Look at the Evolution of DNN](#)

[From Humble Beginnings...](#)

[The Dot-Com Era](#)

[IBuySpy Portal](#)

[ASP.NET](#)

[IBuySpy Portal Forum](#)

[IBuySpy Workshop](#)

[Subscription Fiasco](#)

[Microsoft](#)

[DotNetNuke](#)

[Licensing](#)

[Core Team](#)

[XXL Fork](#)

[Trademarks](#)

[Sponsorship](#)

[Enhancements](#)

[Security Flaw](#)

[DotNetNuke 2.0](#)

[DotNetNuke.com Website](#)

[Provider Model](#)

[Open Source Philosophy](#)
[Stabilization](#)
[Third-Party Components](#)
[Core Team Reorganization](#)
[Microsoft Membership API](#)
[“Breaking” Changes](#)
[Web Hosters](#)
[DotNetNuke 3.0](#)
[Release Schedule](#)
[DotNetNuke Projects](#)
[Intellectual Property](#)
[Marketing](#)
[Microsoft Hosting Program](#)
[Infrastructure](#)
[Branding](#)
[Tech Ed](#)
[Credibility](#)
[Trademark Policy](#)
[ASP.NET 2.0](#)
[Reorganization](#)
[Microsoft Conferences](#)
[DotNetNuke 4.0](#)
[Slashdotted](#)
[Benefactor Program](#)
[Opportunists](#)
[Yin and Yang](#)
[A New Company](#)
[Larry Augustin](#)
[Performance](#)
[DotNetNuke Marketplace](#)
[Free Module Promotion](#)

[Conferences](#)

[Microsoft Valuable Professionals](#)

[Fundraising](#)

[Awards and Accolades](#)

[DotNetNuke OpenForce 07](#)

[SLA Program](#)

[More Fundraising](#)

[CodePlex](#)

[Security Issues](#)

[IP Disputes](#)

[Term Sheets](#)

[DotNetNuke OpenForce 08](#)

[DotNetNuke Professional](#)

[Series A Announcement](#)

[Physical Offices](#)

[DotNetNuke 5.0](#)

[Day of DotNetNuke](#)

[DNN-Europe](#)

[Snowcovered Acquisition](#)

[Telerik Partnership](#)

[Series B](#)

[Open-DocumentLibrary Acquisition](#)

[DotNetNuke Enterprise Edition](#)

[POET Vulnerability](#)

[DotNetNuke.com Overhaul](#)

[Active Modules Acquisition](#)

[Nik Kalyani Leaves DNN Corp](#)

[Cloud. Mobile. Social.](#)

[DotNetNuke 6.0](#)

[DotNetNuke World 2011](#)

[DotNetNuke Gets Social](#)

[Microsoft Azure Partnership](#)

[DNN World 2012](#)

[DotNetNuke 7.0](#)

[iFinity Acquisition](#)

[10-Year Anniversary](#)

[DNN Social](#)

[DotNetNuke.com Hacked](#)

[Rebranding](#)

[DNNCon](#)

[Scott Willhite Moves On](#)

[DNN 7.x Releases](#)

[My Departure from DNN Corp](#)

[Summary](#)

[Chapter 2: Installing DNN Version 7](#)

[What You Need To Install DNN Platform Version 7](#)

[Upgrading the DNN Platform to Version 7](#)

[Getting a Trial Version of Evoq Content](#)

[Common Installation Issues](#)

[Summary](#)

[Chapter 3: DNN Platform Overview](#)

[Core Platform Objects](#)

[Security](#)

[Summary](#)

[Chapter 4: Site Administration](#)

[Wrox.com Code Downloads for this Chapter](#)

[What Is Site Administration?](#)

[Common Administrative Tasks](#)

[Admin Menu Features](#)

[Best Practices for Site Administrators](#)

[Summary](#)

[Chapter 5: Host Administration](#)

[Why Do You Need the Host?](#)

[What Is Host Administration?](#)

[Host Menu Pages](#)

[Additional Host Features on Admin Site Settings](#)

[Additional Host Features on the Control Panel](#)

[Host Options on the Module Actions Menu](#)

[Integrating with a Third-Party Provider](#)

[Summary](#)

[Chapter 6: Modules](#)

[What Is a Module?](#)

[Where Do Modules Live on a Page?](#)

[Adding a Module to a Page](#)

[One Module Across Multiple Pages](#)

[One Module Across Multiple Sites](#)

[Working with Modules](#)

[Where to Get Modules](#)

[Viewing Modules and Extensions](#)

[Installing Modules into DNN](#)

[The Extension Verification System](#)

[In Depth with the HTML Module](#)

[Summary](#)

[Chapter 7: System Architecture](#)

[Patterns and Concepts](#)

[Architectural Overview](#)

[Namespace Overview](#)

[Summary](#)

[Chapter 8: Core DNN APIs](#)

[The CBO Class](#)

[Caching](#)

[Event Logging](#)

[Exception Management](#)

[Scheduler](#)

[Module Interfaces](#)

[Summary](#)

[Chapter 9: Membership Security](#)

[Wrox.com Code Downloads for this Chapter](#)

[DNN Membership Overview](#)

[Membership Provider](#)

[Authentication Providers](#)

[Membership Management Enhancements](#)

[Summary](#)

[Chapter 10: Localization](#)

[Locales in DNN](#)

[Resource Files](#)

[The API](#)

[Localizing Modules](#)

[Summary](#)

[Chapter 11: Search](#)

[History](#)

[Objectives of the New Search Functionality](#)

[Apache Lucene](#)

[Search Architecture](#)

[Platform Features](#)

[Evoq Features](#)

[Administration](#)

[Search Phases](#)

[Module Integration](#)

[Entities](#)

[APIs](#)

[Writing a New Crawler](#)

[Troubleshooting](#)

[Summary](#)

[Chapter 12: URL Management](#)

[The History of DNN URL Schemes](#)

[Understanding URL Structure in DNN](#)

[URL Configuration and Customization](#)

[Summary](#)

[Chapter 13: Beginning Module Development](#)

[Wrox.com Code Downloads for this Chapter](#)

[A Guided Tour of Your Work Environment](#)

[Your Toolbox](#)

[The Environment](#)

[Organizing Your Project](#)

[Module Design Considerations](#)

[About Modules, TabModules, Module Definitions](#)

[A Guestbook Module](#)

[Wrapping It Up](#)

[Summary](#)

[Chapter 14: Developing Modules: User Interfaces](#)

[Wrox.com Code Downloads for this Chapter](#)

[Understanding DNN and Module Interactions](#)

[Dialogs and AJAX Support](#)

[JavaScript, jQuery, and Custom Scripts](#)

[DNN jQuery Plugins](#)

[Implementing Consistent Design](#)

[Summary](#)

[Chapter 15: Developing Modules: Business Logic](#)

[Wrox.com Code Downloads for this Chapter](#)

[Navigating with the DNN API](#)

[Using Common DotNetNuke Controls](#)

[Leveraging Web API](#)

[Controlling Navigation and Module Views](#)

[Summary](#)

[Chapter 16: Developing Modules: Best Practices and Moving Forward](#)

[Wrox.com Code Downloads for this Chapter](#)

[Managing DNN References and Versions](#)

[Managing External Dependencies](#)

[Future-Proofing Data Interactions](#)

[Extension Verification Service \(EVS\)](#)

[Getting Prepared for DNN neXt](#)

[Summary](#)

[Chapter 17: Skinning](#)

[Wrox.com Code Downloads for this Chapter](#)

[Skinning by Today's Standards](#)

[Parts of a DNN Skin](#)

[Skinning Approaches](#)

[Preparing to Create a Skin](#)

[Creating Your First Skin](#)

[Basic Layout](#)

[Document Setup](#)

[Skin Objects](#)

[Navigation](#)

[Creating Alternate Skins](#)

[Creating Containers](#)

[Custom 404 and Pop-up Skins](#)

[Skin Thumbnails](#)

[Creating an Installable Skin Package](#)

[Advanced Skinning Techniques](#)

[Summary](#)

[Chapter 18: Packaging and Distribution](#)

[The New Extensions Model](#)

[Creating New Extensions](#)

[Using the Wizard to Create Packages](#)

[Building Packages with Manifest Files](#)

[Summary](#)

[Chapter 19: Commercial Philosophy](#)

[The Fundamentals](#)

[Technology](#)

[Market Conditions](#)

[Distribution Model](#)

[Branding](#)

[Results](#)

[SUMMARY](#)

[Chapter 20: Evoq Content](#)

[Content Creation](#)

[Permissions, Workflow, and Versioning](#)

[Optimization](#)

[Integrations](#)

[Summary](#)

[Chapter 21: Evoq Engage](#)

[Management Tools](#)

[Community Modules](#)

[Summary](#)

[Chapter 22: The DNN Store](#)

[Buying from the Store](#)

[The Referral Program](#)

[Selling on the Store](#)

[Summary](#)

[Chapter 23: DNN on Microsoft Azure](#)

[Wrox.com Code Downloads for this Chapter](#)

[Azure Deployment Scenarios](#)

[Installing DNN on Azure Websites](#)

[Remote Connections to Azure Websites](#)

[Backing Up Your Azure Website](#)

[Upgrading to a New DNN Version](#)

[Moving an Existing DNN Site to Azure Websites](#)

[Managing and Troubleshooting Your Azure Website](#)

[Summary](#)

[Appendix A: Resources](#)

[Appendix B: System Message Tokens](#)

[Advertisement](#)

[End User License Agreement](#)

List of Illustrations

Chapter 1: An Inside Look at the Evolution of DNN

[Figure 1.1](#)

[Figure 1.2](#)

[Figure 1.3](#)

[Figure 1.4](#)

[Figure 1.5](#)

[Figure 1.6](#)

[Figure 1.7](#)

[Figure 1.8](#)

[Figure 1.9](#)

[Figure 1.10](#)

[Figure 1.11](#)

[Figure 1.12](#)

[Figure 1.13](#)

[Figure 1.14](#)

[Figure 1.15](#)

[Figure 1.16](#)

[Figure 1.17](#)

[Figure 1.18](#)

[Figure 1.19](#)

[Figure 1.20](#)

[Figure 1.21](#)

[Figure 1.22](#)

[Figure 1.23](#)

[Figure 1.24](#)

[Figure 1.25](#)

[Figure 1.26](#)

Chapter 2: Installing DNN Version 7

[Figure 2.1](#)

[Figure 2.2](#)

[Figure 2.3](#)

[Figure 2.4](#)

[Figure 2.5](#)

[Figure 2.6](#)

[Figure 2.7](#)

[Figure 2.8](#)

[Figure 2.9](#)

[Figure 2.10](#)

[Figure 2.11](#)

[Figure 2.12](#)

[Figure 2.13](#)

[Figure 2.14](#)

Chapter 3: DNN Platform Overview

[Figure 3.1](#)

[Figure 3.2](#)

[Figure 3.3](#)

[Figure 3.4](#)

[Figure 3.5](#)

[Figure 3.6](#)

[Figure 3.7](#)

[Figure 3.8](#)

[Figure 3.9](#)

[Figure 3.10](#)

[Figure 3.11](#)

Chapter 4: Site Administration

[Figure 4.1](#)

[Figure 4.2](#)

[Figure 4.3](#)

[Figure 4.4](#)

[Figure 4.5](#)

[Figure 4.6](#)

[Figure 4.7](#)

[Figure 4.8](#)

[Figure 4.9](#)

[Figure 4.10](#)

[Figure 4.11](#)

[Figure 4.12](#)

[Figure 4.13](#)

[Figure 4.14](#)

[Figure 4.15](#)

[Figure 4.16](#)

[Figure 4.17](#)

[Figure 4.18](#)

[Figure 4.19](#)

[Figure 4.20](#)

[Figure 4.21](#)

[Figure 4.22](#)

[Figure 4.23](#)

[Figure 4.24](#)

[Figure 4.25](#)

[Figure 4.26](#)

[Figure 4.27](#)

[Figure 4.28](#)

[Figure 4.29](#)

[Figure 4.30](#)

[Figure 4.31](#)

[Figure 4.32](#)

[Figure 4.33](#)

[Figure 4.34](#)

[Figure 4.35](#)

[Figure 4.36](#)

[Figure 4.37](#)

[Figure 4.38](#)

[Figure 4.39](#)

[Figure 4.40](#)

[Figure 4.41](#)

[Figure 4.42](#)

[Figure 4.43](#)

[Figure 4.44](#)

[Figure 4.45](#)

[Figure 4.46](#)

Chapter 5: Host Administration

[Figure 5.1](#)

[Figure 5.2](#)

[Figure 5.3](#)

[Figure 5.4](#)

[Figure 5.5](#)

[Figure 5.6](#)

[Figure 5.7](#)

[Figure 5.8](#)

[Figure 5.9](#)

[Figure 5.10](#)

[Figure 5.11](#)

[Figure 5.12](#)

[Figure 5.13](#)

[Figure 5.14](#)

[Figure 5.15](#)

[Figure 5.16](#)

[Figure 5.17](#)

[Figure 5.18](#)

[Figure 5.19](#)

[Figure 5.20](#)

[Figure 5.21](#)

[Figure 5.22](#)

[Figure 5.23](#)

[Figure 5.24](#)

[Figure 5.25](#)

[Figure 5.26](#)

[Figure 5.27](#)

[Figure 5.28](#)

[Figure 5.29](#)

[Figure 5.30](#)

[Figure 5.31](#)

[Figure 5.32](#)

Chapter 6: Modules

[Figure 6.1](#)

[Figure 6.2](#)

[Figure 6.3](#)

[Figure 6.4](#)

[Figure 6.5](#)

[Figure 6.6](#)

Chapter 7: System Architecture

[Figure 7.1](#)

Chapter 9: Membership Security

[Figure 9.1](#)

[Figure 9.2](#)

[Figure 9.3](#)

[Figure 9.4](#)

[Figure 9.5](#)

[Figure 9.6](#)

[Figure 9.7](#)

[Figure 9.8](#)

Chapter 10: Localization

[Figure 10.1](#)

[Figure 10.2](#)

[Figure 10.3](#)

[Figure 10.4](#)

[Figure 10.5](#)

[Figure 10.6](#)

[Figure 10.7](#)

[Figure 10.8](#)

Chapter 11: Search

[Figure 11.1](#)

[Figure 11.2](#)

[Figure 11.3](#)

[Figure 11.4](#)

[Figure 11.5](#)

[Figure 11.6](#)

[Figure 11.7](#)

[Figure 11.8](#)

[Figure 11.9](#)

[Figure 11.10](#)

[Figure 11.11](#)

[Figure 11.12](#)

[Figure 11.13](#)

[Figure 11.14](#)

[Figure 11.15](#)

[Figure 11.16](#)

[Figure 11.17](#)

[Figure 11.18](#)

[Figure 11.19](#)

[Figure 11.20](#)

[Figure 11.21](#)

[Figure 11.22](#)

[Figure 11.23](#)

[Figure 11.24](#)

[Figure 11.25](#)

[Figure 11.26](#)

[Figure 11.27](#)

[Figure 11.28](#)

[Figure 11.29](#)

[Figure 11.30](#)

[Figure 11.31](#)

[Figure 11.32](#)

[Figure 11.33](#)

[Figure 11.34](#)

[Figure 11.35](#)

[Figure 11.36](#)

[Figure 11.37](#)

[Figure 11.38](#)

[Figure 11.39](#)

[Figure 11.40](#)

[Figure 11.41](#)

[Figure 11.42](#)

[Figure 11.43](#)

[Figure 11.44](#)

Chapter 12: URL Management

[Figure 12.1](#)

[Figure 12.2](#)

[Figure 12.3](#)

[Figure 12.4](#)

[Figure 12.5](#)

[Figure 12.6](#)

[Figure 12.7](#)

Chapter 13: Beginning Module Development

[Figure 13.1](#)

[Figure 13.2](#)

[Figure 13.3](#)

[Figure 13.4](#)

[Figure 13.5](#)

[Figure 13.6](#)

[Figure 13.7](#)

[Figure 13.8](#)

[Figure 13.9](#)

[Figure 13.10](#)

[Figure 13.11](#)

[Figure 13.12](#)

[Figure 13.13](#)

[Figure 13.14](#)

[Figure 13.15](#)

[Figure 13.16](#)

[Figure 13.17](#)

[Figure 13.18](#)

[Figure 13.19](#)

Chapter 14: Developing Modules: User Interfaces

[Figure 14.1](#)

[Figure 14.2](#)

Chapter 15: Developing Modules: Business Logic

[Figure 15.1](#)

[Figure 15.2](#)

[Figure 15.3](#)

[Figure 15.4](#)

[Figure 15.5](#)

Chapter 16: Developing Modules: Best Practices and Moving Forward

[Figure 16.1](#)

[Figure 16.2](#)

[Figure 16.3](#)

Chapter 17: Skinning

[Figure 17.1](#)

[Figure 17.2](#)

[Figure 17.3](#)

[Figure 17.4](#)

[Figure 17.5](#)

[Figure 17.6](#)

[Figure 17.7](#)

[Figure 17.8](#)

[Figure 17.9](#)

[Figure 17.10](#)

[Figure 17.11](#)

[Figure 17.12](#)

Chapter 18: Packaging and Distribution

[Figure 18.1](#)

[Figure 18.2](#)

[Figure 18.3](#)

[Figure 18.4](#)

[Figure 18.5](#)

[Figure 18.6](#)

[Figure 18.7](#)

Chapter 20: Evoq Content

[Figure 20.1](#)

[Figure 20.2](#)

[Figure 20.3](#)

[Figure 20.4](#)

Chapter 21: Evoq Engage

[Figure 21.1](#)

[Figure 21.2](#)

[Figure 21.3](#)

[Figure 21.4](#)

[Figure 21.5](#)

[Figure 21.6](#)

[Figure 21.7](#)

[Figure 21.8](#)

[Figure 21.9](#)

Chapter 23: DNN on Microsoft Azure

[Figure 23.1](#)

[Figure 23.2](#)

[Figure 23.3](#)

[Figure 23.4](#)

[Figure 23.5](#)

[Figure 23.6](#)

[Figure 23.7](#)

[Figure 23.8](#)

[Figure 23.9](#)

[Figure 23.10](#)

[Figure 23.11](#)

[Figure 23.12](#)

[Figure 23.13](#)

[Figure 23.14](#)

[Figure 23.15](#)

[Figure 23.16](#)

[Figure 23.17](#)

[Figure 23.18](#)

[Figure 23.19](#)

[Figure 23.20](#)

[Figure 23.21](#)

[Figure 23.22](#)

[Figure 23.23](#)

[Figure 23.24](#)

[Figure 23.25](#)

[Figure 23.26](#)

[Figure 23.27](#)

[Figure 23.28](#)

[Figure 23.29](#)

[Figure 23.30](#)

[Figure 23.31](#)

[Figure 23.32](#)

[Figure 23.33](#)

[Figure 23.34](#)

[Figure 23.35](#)

[Figure 23.36](#)

[Figure 23.37](#)

[Figure 23.38](#)

[Figure 23.39](#)

[Figure 23.40](#)

[Figure 23.41](#)

[Figure 23.42](#)

[Figure 23.43](#)

[Figure 23.44](#)

[Figure 23.45](#)

[Figure 23.46](#)

[Figure 23.47](#)

[Figure 23.48](#)

List of Tables

Chapter 4: Site Administration

[Table 4.1 Digital Asset Manager Control Panel](#)

[Table 4.2 Folder Menu Options](#)

[Table 4.3 DNN Platform Folder Providers](#)

[Table 4.4 Folder Permissions](#)

[Table 4.5 Pages Module Context Menu Options](#)

[Table 4.6 Built-in Security Roles](#)

[Table 4.7 Edit Security Roles Settings](#)

[Table 4.8 Page Management Settings](#)

[Table 4.9 User Registration Options](#)

[Table 4.10 Display Name Format Tokens](#)

Chapter 8: Core DNN APIs

[Table 8.1 The CBO Hydration Methods](#)

[Table 8.2 Selected Public Methods in DataCache](#)

[Table 8.3 Properties of the CacheItemArgs Class](#)

[Table 8.4 The LogController Class](#)

[Table 8.5 Properties of the LogInfo Class](#)

[Table 8.6 Helper Methods in the Exceptions Class](#)

[Table 8.7 Members of the IModuleControl Interface](#)

[Table 8.8 Members of the ModuleAction Class](#)

Chapter 10: Localization

[Table 10.1 Localization Methods](#)

[Table 10.2 GetString Parameters](#)

[Table 10.3 GetSystemMessage Parameters](#)

[Table 10.4 Objects Available for Token Replacement](#)

[Table 10.5 Wrapping a String with a Web Control](#)

[Table 10.6 Adding a Resource Key](#)

[Table 10.7 Default Localized Attributes](#)

Chapter 11: Search

[Table 11.1 Site Crawler's Tasks](#)

[Table 11.2 Steps in Content Indexing](#)

[Table 11.3 `SearchType` Properties](#)

[Table 11.4 `SearchDocument` Properties](#)

[Table 11.5 Property Boost Levels](#)

[Table 11.6 `SearchQuery` Properties](#)

[Table 11.7 `SearchResult` Properties](#)

[Table 11.8 Indexing APIs](#)

[Table 11.9 Administration APIs](#)

[Table 11.10 Lucene Internal APIs](#)

Chapter 12: URL Management

[Table 12.1 URL Mode Comparison](#)

[Table 12.2 DNN URL Types](#)

[Table 12.3 Create URL Fields](#)

[Table 12.4 Rewriting Test Result Fields](#)

[Table 12.5 Installation-Level Configuration Options](#)

[Table 12.6 Example Site Aliases](#)

[Table 12.7 Site Alias Configuration](#)

[Table 12.8 Site Alias Values](#)

[Table 12.9 SSL Settings](#)

[Table 12.10 Evoq Advanced URL Settings](#)

[Table 12.11 Advanced URL Management Options](#)

[Table 12.12 Creating DNN Page URLs](#)

[Table 12.13 Common HTTP Response Codes](#)

[Table 12.14 URL Debug Response Header Data](#)

Chapter 13: Beginning Module Development

[Table 13.1 Major Directories in a New DNN Installation](#)

[Table 13.2 Major DNN Releases between 2011 and 2014](#)

[Table 13.3 View.ascx.resx Entries](#)

[Table 13.4 Edit.ascx.resx entries](#)

Chapter 15: Developing Modules: Business Logic

[Table 15.1 Common Methods](#)

[Table 15.2 Helpful Host Object Properties](#)

[Table 15.3 Helpful PortalSettings Properties](#)

[Table 15.4 RegisterRoutes Method Parameters](#)

Chapter 17: Skinning

[Table 17.1 CSS File Priorities](#)

[Table 17.2 JavaScript File Priorities](#)

Chapter 18: Packaging and Distribution

[Table 18.1 Package Settings Attributes](#)

[Table 18.2 Configuration Options for Module Extensions](#)

[Table 18.3 Module Control Definition Attributes](#)

[Table 18.4 Wrox.Suggestion Module Definitions Module Controls](#)

[Table 18.5 Configuration Options for Skin Object Extensions](#)

[Table 18.6 Configuration Options for Authentication System Extensions](#)

[Table 18.7 Package Elements and Attributes](#)

[Table 18.8 Node Action Types](#)

Appendix A: Resources

[Table A.1 Developer Tools](#)

[Table A.2 Popular Extensions](#)

[Table A.3 dnnsoftware.com resources](#)

Appendix B: System Message Tokens

[Table B.1 Standard HostSettings Properties](#)

[Table B.2 Standard PortalSettings Properties](#)

[Table B.3 Standard UserInfo Properties](#)

[Table B.4 Standard UserMembership Properties](#)

[Table B.5 Standard UserProfile Properties](#)

Introduction

DNN7 is an open source CMS platform developed using Microsoft's ASP.NET technology. It can be used by end users and administrators as an advanced content management system for the creation and management of dynamic websites. It can also be used by software developers as a platform for building sophisticated ASP.NET web applications.

Who This Book Is For

This book is suitable for all audiences. It contains content for end users and administrators who are interested in learning how to utilize the software to create and maintain advanced websites. It also contains technical content for software developers and web designers who want to build custom extensions or skins for the platform.

What This Book Covers

This book is focused on the functionality that is present in the DNN7 product editions. Users of earlier product editions may also find this book useful, as many concepts and features that existed in previous versions have remained consistent over time. This book also includes a complete history of the open source project and business model, as well as dedicated coverage of the features and functionality of the commercial Evoq solutions.

How This Book Is Structured

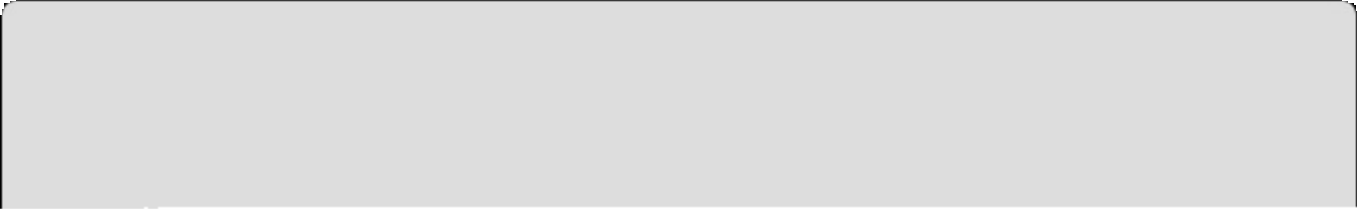
This book is logically divided into four sections. The first section explores the history of the open source project, explains how to download and install the product, and describes how to manage and administrate a DNN website. The second section explores the application architecture and its major application programming interfaces (APIs). The third section of the book demonstrates how you can extend the platform by developing and distributing modules and skins that integrate seamlessly with your DNN website. The final section explains the open source business model and describes the advanced features of the commercial Evoq solutions, which are built on top of the DNN platform.

What You Need to Use This Book

To utilize DNN, you need any of Windows 2008/2012 Server, Windows 7, or Windows 8 (the latter two for development only). This book relies on SQL Server as the database provider. You must have access to SQL Server 2008 or above or an equivalent version of SQL Express Edition (development only) on the same machine or remotely over the network. To participate in the development chapters, you will need Visual Studio 2010 or above or an equivalent version of the free Visual Studio Express or Visual Web Developer. DNN7 runs on the .NET Framework 4.0 and above.

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.



WARNING

Boxes like this one hold important, not-to-be forgotten information that is directly relevant to the surrounding text.



NOTE

Notes, tips, hints, tricks, and asides to the current discussion are offset like this.

As for styles in the text:

- We *italicize* new terms and important words when we introduce them.
- We present keyboard strokes like this: Ctrl+A.
- We show URLs and code within the text like so: `persistence.properties`.
- We present code in the following way:

We use a monofont type with no highlighting for most code examples. We use **bold** to emphasize code that is particularly important in the present context or to show changes from a previous code snippet.

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually, or to use the source code files that accompany the book. All the source code used in this book is available for download at www.wrox.com. Specifically for this book, the code download is on the Download Code tab at:

www.wrox.com/go/prodnn7

You can also search for the book at www.wrox.com by ISBN (the ISBN for this book is 978-1-118-85084-8) to find the code. And a complete list of code downloads for all current Wrox books is available at

www.wrox.com/dynamic/books/download.aspx.

Each chapter that contains downloadable code or other support files will indicate so in the text at the beginning of the chapter.

Most of the code on www.wrox.com is compressed in a .ZIP, .RAR archive or similar archive format appropriate to the platform. Once you download the code, just decompress it with an appropriate compression tool.



NOTE

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-1-118-85084-8.

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at www.wrox.com/dynamic/books/download.aspx to see the code available for this book and all other Wrox books.

Errata

Every effort is made to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, your feedback is welcome. By sending in errata, you may save other readers hours of frustration and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to www.wrox.com and locate the title using the Search box or one of the title lists. Then, on the book's detail page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at www.wrox.com/misc-pages/booklist.shtml.

If you don't spot “your” error on the Book Errata page, go to www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. After the information is checked, a message is posted to the book's errata page, and the problem is fixed in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At p2p.wrox.com you can find a number of different forums that can help you not only as you read this book but also as you develop your own applications. To join the forums, follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you want to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.



NOTE

You can read messages in the forums without joining P2P, but to post your own messages, you must join.

After you join, you can post new messages and respond to messages other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum e-mailed to you, click the **Subscribe to This Forum** icon by the forum name in the forum listing.

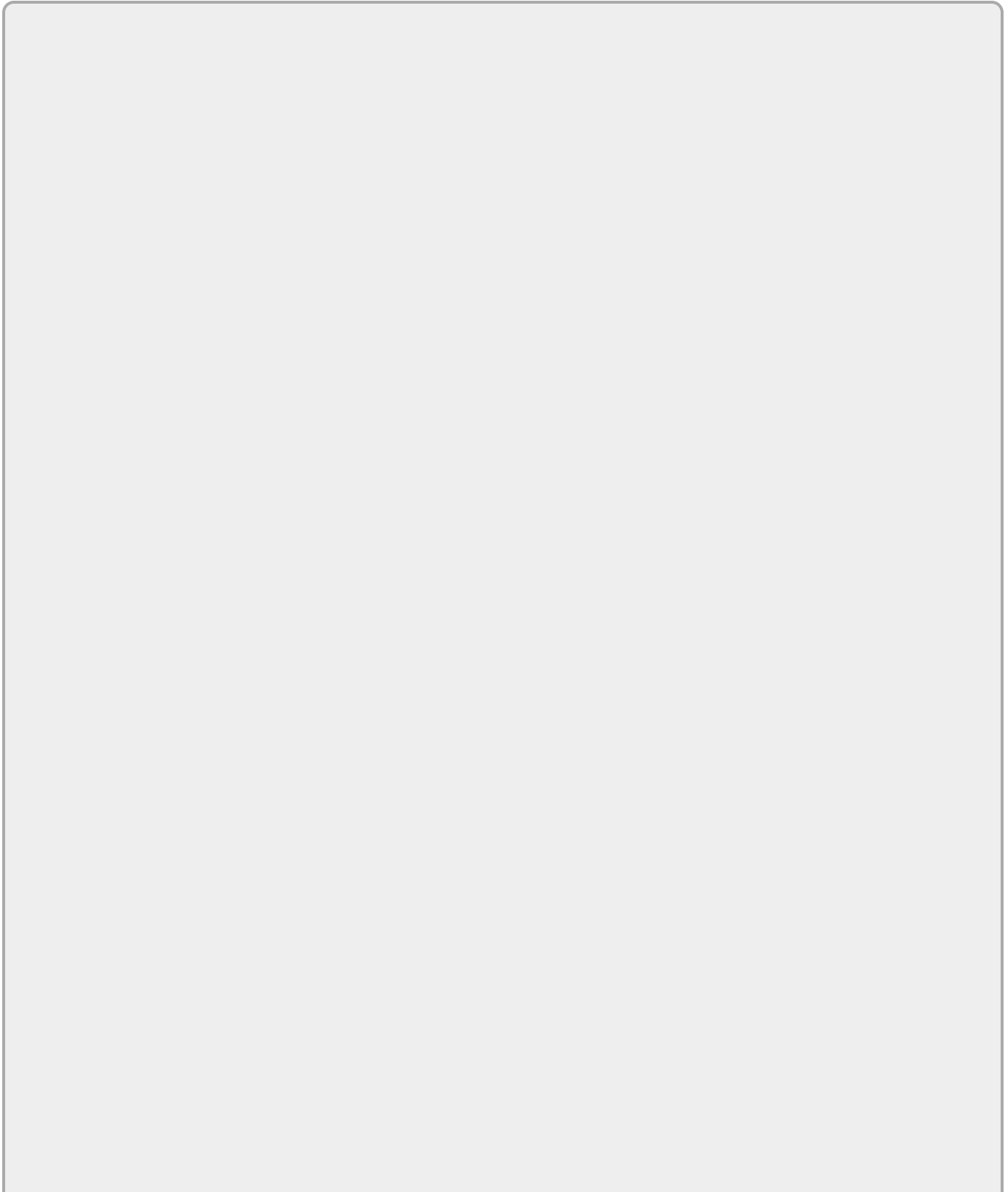
For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

Chapter 1

An Inside Look at the Evolution of DNN

By Shaun Walker

DotNetNuke Creator and DNN CORP Cofounder



What You Will Learn in this Chapter

- Following the behind-the-scenes story of one of the most-loved community technology projects
- Discovering the legal, licensing, and IP business realities of open source software
- Understanding the branding evolution of DotNetNuke to DNN
- Exploring how open source and commercial business models can work together
- Learning about start-ups, strategic marketing, partnerships, acquisitions, and venture capital funding

As much as DNN is an open source software application written for the Microsoft ASP.NET platform, it is also an online community with developers, end users, vendors, and volunteers—all working together collaboratively in a rich and diverse ecosystem. This chapter attempts to capture the essence of the project, expose its humble beginnings, provide insight into its evolution, and document its many achievements, but not shy away from some of the hard lessons learned in the process. The lifeblood of any community is its people; therefore, it is a distinct honor and privilege to be able to share some of the emotion and passion that have gone into the DNN project so that you may be able to establish a personal connection with the various stakeholders, which may ultimately motivate you to join this vibrant ecosystem.

From Humble Beginnings...

In 1979, when I was 9 years old, my family relocated from Kelowna, British Columbia, Canada, to Ashcroft, a tiny community in the south-central interior of British Columbia with a population of approximately 1,500 people. We relocated with a grand vision—to start a commercial vineyard. My grandfather had owned a vineyard in Kelowna and he had sold it with the idea that the hot arid climate in Ashcroft, combined with cheap abundant land, would be a perfect environment for him and his children to establish a large, successful vineyard.

My mother and father, grandparents, and two uncles bought approximately 200 acres of sagebrush-covered land about 30 minutes outside Ashcroft, an area known as Basque Siding that was only accessible by navigating 5 miles of unpaved roads. We each had our own 50-acre parcel of land but the family all worked together to establish the infrastructure to develop our vineyards. We installed power, irrigation systems, cleared land, and built houses. Then we planted seedling grapes, and grew alfalfa and other crops to provide some initial income, as a vineyard takes five years before it reaches full production. We were extremely self-sufficient and raised our own cows, pigs, turkeys, and chickens, as well as our own fruits and vegetables.

When I was 12 years old we visited my cousins in Kelowna and I was introduced to the Commodore VIC-20 for the first time. My cousins were using it to play games but my parents clearly saw my fascination in this little machine. Money was scarce, so I am not sure what ultimately motivated their decision, but they decided to purchase a base model VIC-20, which came with an integrated keyboard, a cassette tape drive, and a user manual. They also had one stipulation—the only games I could play were games I created myself.

So I spent a lot of time typing BASIC code into the computer and storing the programs on cassette tape. My parents got me a subscription to *COMPUTE!* magazine, which provided source code listings for more advanced games. Pretty soon I started to recognize the patterns and techniques required to write programs, and I started building my own applications. Living on a remote farm created a perfect environment for investing myself in computers, as there were few distractions—I was either outside working in the vineyard (pruning, weeding, picking, cultivating) or I was inside the house typing code. My two younger brothers were more than happy to play the games that I

created for them.

When I saw the movie *War Games* in 1983 starring Mathew Broderick, I really got excited about the potential of computers as more than just standalone devices. In the movie, the character played by Broderick hacks into the NORAD super computer nicknamed “Joshua” using a backdoor password and mistakenly invokes Global Thermonuclear War. In the climax, the supercomputer is tricked into playing Tic-Tac-Toe against itself until it reaches a draw and declares that “the only winning move is not to play.” After watching this movie numerous times I was convinced that I wanted to be a “hacker” and I sent off handwritten letters to many of the vendors listed in *COMPUTE!* magazine asking how I could become a programmer.

In the summer of 1983 we took a long road trip to Disneyland in California and spent some time visiting my uncle in the Bay Area. During this trip my parents finally caved in to my demands for a computer upgrade. The Commodore 64 was a large enhancement over the VIC-20, and we were able to get a good deal on a Commodore 64 package, a 1702 color monitor, and a 1541 floppy disk drive. I could not wait to get home and plug in these amazing new devices.

The following winter my family suffered a significant setback. Some exceptionally cold weather killed all of our grape plants, which ended my parents' dream of operating a large commercial vineyard. They did not have the resources to replant the vineyard, so they both had to work traditional jobs while continuing to operate the farm by selling fruits and vegetables to the local markets to try to make ends meet. This was very hard work, and without any opportunity for vacations as the farm demanded the family's full attention almost year round. The entire family pitched in to keep the farm afloat as my parents tried valiantly to preserve their investment.

When I reached high school, I was allowed to participate in an accelerated learning program that allowed me to take computer science courses that were two grade levels higher than my current grade. This exposed me to IBM PCs and Apple II computers and some new programming languages. I loved the challenge of solving problems and got a lot of satisfaction out of being able to tell the computer to follow my specific instructions. I had found my passion, and I knew at that point that my future career would involve software development. This was well before the Internet was invented—in fact, my household was still on a party line for phone service that we shared with four other families. It was also well before graphical user interfaces, cell phones,

and other technology that we take for granted now.

When I was 17 years old my family had another major setback. My father collapsed at work and was rushed to the hospital in Kamloops. The doctor discovered that he had suffered a brain aneurysm—at the age of 37. The high stress of trying to keep the farm afloat was identified as one of the potential causes for the aneurysm, which finally prompted my parents to make the decision to cut their losses and get rid of the farm immediately. Because there were no interested buyers, my parents decided to walk away from it with only two vehicles and enough money to put a minimum down payment on a small house in Ashcroft. Financially, this was an extremely challenging time for the family.

As ironic as it sounds, moving into Ashcroft was like moving into a big city, as previously it required a 30–45 minute drive to get even basic necessities such as groceries. I got a job as a dishwasher at one of the few local restaurants and worked 20–30 hours a week during my final years of high school. In addition to working, I also played competitive ice hockey on a rep team and probably could have pursued a scholarship if I had taken it more seriously. I graduated with honors in 1989 from Ashcroft Secondary School. My graduating class had only 30 students, most of whom left town almost immediately to find suitable work.

The opportunities for scholarships in a small town were rather slim, and because my parents were in the process of starting over financially, they did not have the resources to help me with my education. So I decided to take a year off from school and focus on working and saving some money. I moved to Sunnyvale, California, to work in residential construction with my uncle. I got an apartment with two friends from high school. This was a huge culture shock moving from a town of 1,500 people to an apartment building in Silicon Valley that had almost 1,500 people. I grew up quickly that year, taking care of myself and learning about life and responsibilities. But I knew after 1 year that I needed to get back to my true passion of software development.

I decided to go to Okanagan College in Kelowna and take a 2-year diploma program. I would have considered taking a university degree program, but the cost was too prohibitive for my meager financial means at the time (I was denied access to student loans because of a technicality in the application process). I got a job at the Keg Restaurant in Kelowna and worked full-time as a dishwasher, cook, and server while I was going to college so that I could make ends meet. The Computer Information Systems program had a co-op

option that allowed me to get some work experience at a small software company in Mission, British Columbia, which was a provider of financial products for school districts and municipalities. This was a great launch pad for my career as a software developer, as I was given a lot of freedom and authority to build and enhance enterprise products. I graduated from college in late 1992 with a CIS diploma and entered the workforce as a software developer.

One thing that my family always emphasized to me was to focus on solving problems in a repeatable manner. It is not a good use of time to have to reinvent the wheel each time you encounter a problem—it is better to build a solution that you can utilize over and over. In my early software career this served me well, as it is a mind-set that is essential to building software products. And going beyond single-use applications, my passion was on building tools that could be utilized to build many types of software products. Essentially, this involved creating libraries or frameworks that could be the building blocks for larger applications. Between 1993 and 2001 I worked in a variety of private and public software product environments, creating many tools and frameworks in different languages, in different environments, and on different hardware platforms.

The Dot-Com Era

In 2001–2002, I was working for a medium-sized software consulting company in Abbotsford, British Columbia, that was providing outsourced software development services to a variety of large U.S. clients specializing primarily in e-learning initiatives. The internal push was to achieve CMM 3.0 on a fairly aggressive schedule so that we could compete with the emerging outsourcing powerhouses from India and China. As a result there was an incredible amount of focus on process and procedure and somewhat less focus on the technical aspects of software engineering. Because the majority of the client base was interested in the J2EE platform, the company primarily hired resources with Java skills—leaving me with my legacy Microsoft background to assume more of an internal-development and project-management role. The process improvement exercise consumed a lot of time and energy for the company, attempting to better define roles and responsibilities and ensuring proper documentation throughout the project life cycle. Delving into CMM and the PMBOK were great educational benefits for me—skills that would prove to be invaluable in future endeavors. Ultimately, the large U.S. clients decided to test the overseas outsourcing options anyway, which resulted in severe downsizing for the company. It was during these tumultuous times that I recognized the potential of the newly released .NET Framework (beta) and decided that I would need to take my own initiative to learn this exciting new platform to preserve my long-term employment outlook.

For a number of years, I had been maintaining an amateur hockey statistics application as a sideline hobby business. The client application was written in Visual Basic 6.0 with a Microsoft Access backend and I augmented it with a simplistic web publishing service using Active Server Pages 3.0 and SQL Server 7.0. However, better integration with the World Wide Web was quickly becoming the most highly requested enhancement, and I concluded that an exploration into ASP.NET was the best way to enhance the application and at the same time acquire the skills necessary to adapt to the changing landscape. My preferred approach to learning new technologies is to experience them firsthand rather than through theory or traditional education. It was during a Microsoft Developer Days conference in Vancouver, British Columbia, in 2001 that I became aware of a reference application known as the IBuySpy Portal.

IBuySpy Portal

Realizing the educational value of sample applications, Microsoft built a number of source projects that were released with the .NET Framework 1.0 Beta to encourage developers to cut their teeth on the new platform. These projects included full source code and a liberal End User License Agreement (EULA), which provided nearly unrestricted usage. Microsoft co-developed the IBuySpy Portal with Vertigo Software and promoted it as a “best practice” example for building applications in the new ASP.NET environment. Despite its obvious shortcomings, the IBuySpy Portal had some strong similarities to both the Microsoft SharePoint Portal and Content Management Server, as well as other open source CMS applications on the Linux/Apache/mySQL/PHP (LAMP) platform. The portal allowed you to create a completely dynamic website consisting of an unlimited number of virtual “tabs” (pages). Each page had a standard header and three content panes—a left pane, middle pane, and right pane (a standard layout for most portal sites). Within these panes, the administrator could dynamically inject “modules”—essentially mini-applications for managing specific types of web content. The IBuySpy Portal application shipped with six modules designed to cover the most common content types (announcements, links, images, discussions, html/text, and XML) as well as a number of modules for administrating the portal site. As an application framework, the IBuySpy Portal (see [Figure 1.1](#)) provided a mechanism for managing users, roles, permissions, tabs, and modules. With these basic services, the portal offered just enough to whet the appetite of many aspiring ASP.NET developers.

ASP.NET Portal - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites History

Address http://localhost/PortalV6/DesktopDefault.aspx

Links Using Validation Class Browser (Local) aspplus Amazon.com Pure ASP.NET EraServer ProjectWeb Class Browser (Trinity)

Welcome rfair@asppages.com | Portal Home | Portal Documentation | Logoff

IBuySpy Portal


Home Employee Info Product Info Discussions About the Portal Admin

Powered by ASP.NET

Quick Launch [Add Link](#)

- [ASP.NET Site](#)
- [GoDotNet.com](#)
- [ASP.NET on MSDN](#)
- [QuickStart Samples](#)

Welcome to the IBuySpy Portal [Edit](#)




Welcome to the **IBuySpy Portal**, the Intranet Home for IBuySpy's corporate employees. This site serves as the hub application for IBuySpy's internal operations. It provides online news, event and sales information, along with interactive discussion forums and employee contact information. In a nutshell, everything needed to maintain and run the fast-growing IBuySpy commercial empire.

Feel free to browse the site and explore. Sign in to obtain edit access to different modules within the framework, as well as view the restricted sections of the site.

This Week's Special [Edit](#)

The QLT2112 **Document Transportation System** is on special this week to clear an overstock. Purchasers of the P38 Escape Vehicle (Air) receive one free.



News and Features [Add New Announcement](#)

Q4 Sales Rise 200% Over Last Year
 IBuySpy online sales for the crucial fourth quarter of last year rose nearly 200% over the previous year, despite a lackluster holiday sales overall. [read more...](#)

Upcoming Events [Add New Event](#)

Spy-o-Rama
This Saturday, usual secret time and place...
 It's back! The premier regional swap meet for spy paraphernalia of every description. Shop early for some amazing bargains.

Dark Ops Sock Hop
Saturday, 8pm to 7, Dark Ops Cafe
 Back by popular demand! Practice your surveillance of the opposite sex, and dance some too. Great opportunity for a brush pass!

Top Movers [Edit](#)

Product Category	Revenue (Millions)	Growth
Travel	60	8
Communications	45	-7.8
Deception	10	9
Munitions	4	-3

Figure 1.1

ASP.NET

The second critical item that Microsoft delivered at this point in time was a community forums page on the www.asp.net website (see [Figure 1.2](#)). This forum provided a focal point for Microsoft developers to meet and collaborate on common issues in an open, moderated environment. Prior to the release of the forums on www.asp.net, there was a real void in terms of Microsoft community participation in the online or global sphere, especially when compared to the excellent community environments on other platforms.

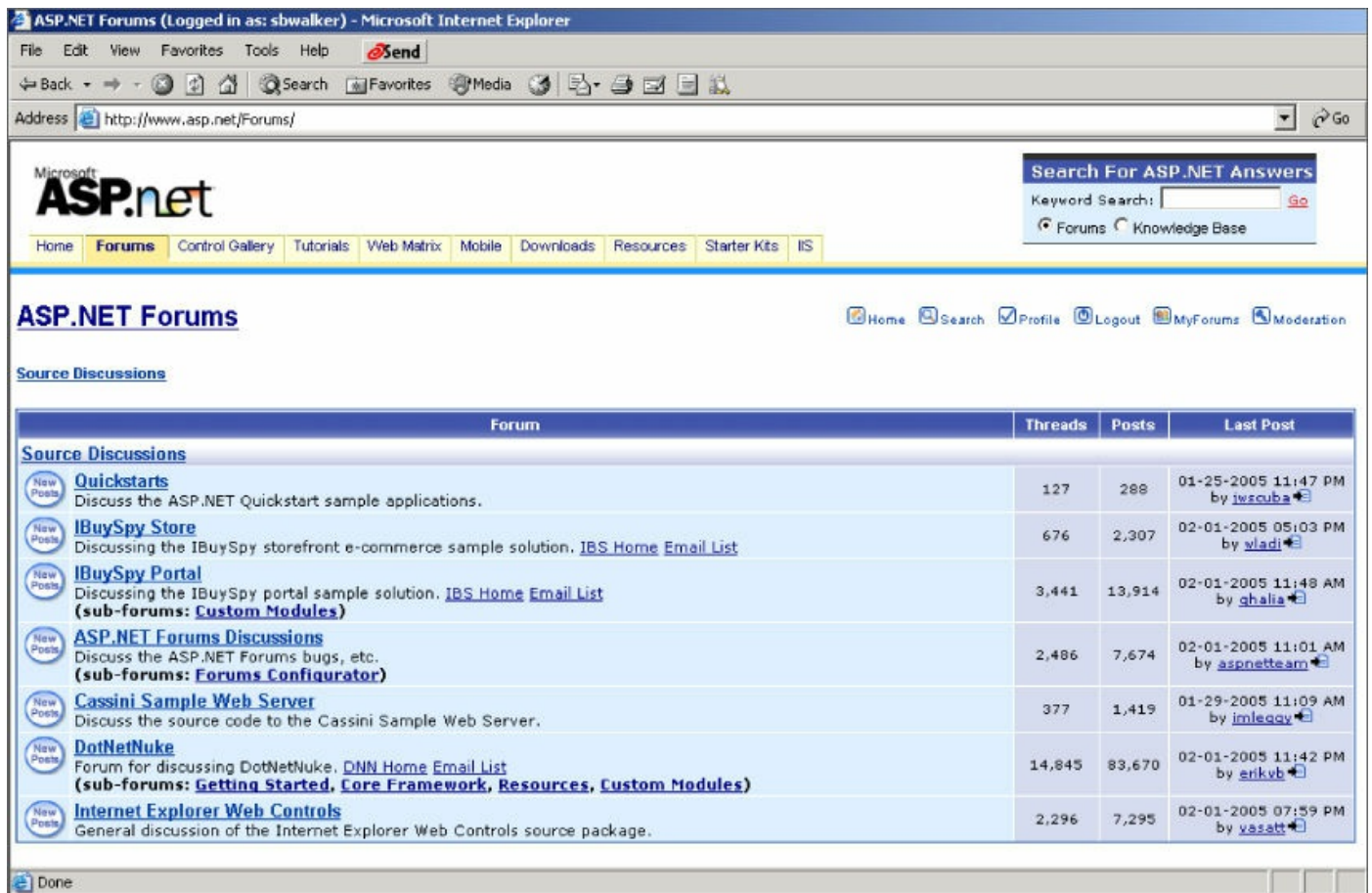


Figure 1.2

One discussion forum on the www.asp.net site was dedicated to the discussion of the IBuySpy Portal application, and it soon became a hotbed for developers to discuss their enhancements, share source code enhancements, and debate IT politics. I became involved in this forum early on and gradually increased my community participation as my confidence in ASP.NET and the IBuySpy Portal application grew.

To appeal to the maximum number of community stakeholders, the IBuySpy

Portal was available in a number of different source-code release packages. There were VB.NET and C#.NET language versions, each containing its own VS.NET and SDK variants. Although Microsoft was aggressively pushing the newly released C# language, I did not feel a compelling urge to abandon my familiar Visual Basic roots. In addition, my experience with classic ASP 3.0 allowed me to conclude that the new code-behind model in VS.NET was far superior to the inline model of the SDK. As luck would have it, I was able to get access to Visual Studio.NET through my employer. So as a result, I moved forward with the VB.NET/VS.NET version as my baseline framework. This decision ultimately proved to be extremely important in terms of community acceptance, as I explain later.

When I first started experimenting with the IBuySpy Portal application, I had some specific objectives in mind. To support amateur sports organizations, I had collected a comprehensive set of end-user requirements based on actual client feedback. However, after evaluating the IBuySpy Portal functionality, it quickly became apparent that some significant enhancements were necessary if I hoped to achieve my goals. My early development efforts, although certainly not elegant or perfectly architected, proved that the IBuySpy Portal framework was highly adaptable for building custom applications and could be successfully used as the foundation for my amateur sports hosting application.

The most significant enhancement I made to the IBuySpy Portal application during these early stages was a feature that is now referred to as “multi-portal” or “site virtualization.” Effectively, this was a fundamental requirement for my amateur sports hosting model. Organizations wanted to have a self-maintained website, but they also wanted to retain their individual identity. A number of vendors emerged with semi-self-maintained web applications, but nearly all of them forced the organization to adopt the vendor's identity (that is, www.vendor.com/clientname rather than www.clientname.com). Although this may seem like a trivial distinction for some, it has some major effects in terms of brand recognition, site discovery, search engine ranking, and so on. The IBuySpy Portal application already partitioned its data by portal (site), and it had a field in the Portals database table named PortalAlias that was a perfect candidate for mapping a specific domain name to a portal. It was as if the original creators (Microsoft and Vertigo) considered this use case during development but did not have enough time to complete the implementation, so they simply left the “hook”

exposed for future development. I immediately saw the potential of this concept and implemented some logic that allowed the application to serve up custom content based on domain name. Essentially, when a web request was received by the application, it would parse the domain name from the URL and perform a lookup on the PortalAlias field to determine the content that should be displayed. This site virtualization capability would ultimately become the “killer” feature that would allow the application to achieve immediate popularity as an open source project.

Over the next 8 to 10 months, I continued to enhance and refactor the IBuySpy Portal application as I created my own custom implementation (now code-named SportsManager.Net). I added numerous features to improve the somewhat limited portal administration and content management aspects. At one point, I enlisted the help of another developer, John Lucarino, and together we steadily improved the framework using whatever spare time we were able to invest. Unfortunately, because all of this was going on outside of regular work hours, there was little time that could be focused on building a viable commercial venture. So at the end of 2002, it soon became apparent that we did not have enough financial backing or a business model to take the amateur sports venture to the next level. This brought the commercial nature of the endeavor under scrutiny. If the commercial intentions were not going to succeed, I at least wanted to feel that my efforts were not in vain. This forced me to evaluate alternative noncommercial uses of the application. Coincidentally, I had released the source code for a number of minor application enhancements to the www.asp.net community forum during the year, and I began to hypothesize that if I abandoned the amateur sports venture altogether, it was still possible that my efforts could benefit the larger ASP.NET community.

The fundamental problem with the IBuySpy Portal community was the fact that there was no central authority in charge of managing its growth. Although Microsoft and Vertigo developed the initial code base, there was no public commitment to maintain or enhance the product in any way. Basically the product was a static implementation, frozen in time, an evolutionary dead-end. However, the IBuySpy Portal EULA was extremely liberal, which meant that developers were free to enhance, license, and redistribute the source code in an unrestricted manner. This led to many developers creating their own customized versions of the application, sometimes sharing discrete patches with the general community, but more often keeping their

enhancements private, revealing only their public-facing websites for community recognition (one of the most popular threads at this time was titled “Show me your Portal”). In hindsight, I really don't understand what each developer was hoping to achieve by keeping his enhancements private. Most probably thought there was a commercial opportunity in building a portal application with a richer feature set than their competitors. Or perhaps individuals were hoping to establish an expert reputation based on their public-facing efforts. Either way, the problem was that this mind-set was really not conducive to building a community but rather to fragmenting it—a standard trap that tends to consume many things on the Microsoft platform. The concept of sharing source code in an unrestricted manner was really a foreign concept, which is obviously why nobody thought to step forward with an organized open source plan.

I have to admit I had a limited knowledge of the open source philosophy at this point because all of my previous experience was in the Microsoft community. However, there was chatter in the forums at various times regarding the organized sharing of source code, and there was obviously some interest in this area. The concept of incorporating the best enhancements into a rapidly evolving open source application made a lot of sense because it benefited the entire community and created a wealth of opportunities for everyone. Coincidentally, a few open source projects had recently emerged on the Microsoft platform to imitate some of the more successful open source projects in the LAMP community. In evaluating my amateur sports application, I soon realized that nearly all of my enhancements were generic enough that they could be applied to nearly any website—they were not sports-related whatsoever. I concluded that I should release my full application source code to the ASP.NET community as a new open source project. So, as a matter of fact, the initial decision to open source what would eventually become DotNetNuke happened more out of frustration of not achieving my commercial goals rather than true philanthropic intentions.

IBuySpy Portal Forum

On December 24, 2002, I released the full open source application by creating a simple website with a Zip file for download. The lack of foresight of what this would become was extremely evident when you consider the casual nature of this original release. However, as luck would have it, I did do a few things right. First, I thought I should leverage the IBuySpy brand in my own open source implementation so that it would be immediately obvious that the code base was a hybrid of the original IBuySpy Portal application, an application with widespread recognition in the Microsoft community. The name I chose was IBuySpy Workshop because it seemed to summarize the evolution of the original application. Rather than assume individual responsibility for the project, I released IBuySpy Workshop as a product of Perpetual Motion Interactive Systems Inc., my personal consulting company. Ironically, I did not even have the domain name resolution properly configured for www.ibuyspyworkshop.com when I released (the initial download links were based on an IP address, <http://65.174.86.217/ibuyspyworkshop>). The second thing I did right was to require people to register on my website before they were able to download the source code. This allowed me to track the actual interest in the application at a more granular level than simply by the total number of downloads. Third, I publicized the availability of the application in the IBuySpy Portal Forum on www.asp.net (see [Figure 1.3](#)). This particular forum was extremely popular at this time; and as far as I know, nobody had ever released anything other than small code snippet enhancements for general consumption. The original post was made on Christmas Eve, December 24, 2002, which had excellent symbolism in terms of the application being a gift to the community.

Author	Thread: Merry Christmas! - Ho! Ho! Ho! - FREE IBS Multi-Portal Implementation
sbwalker	<p>Merry Christmas! - Ho! Ho! Ho! - FREE IBS Multi-Portal Implementation Posted: 12-24-2002 02:47 PM</p> <p>A FREE multi-portal implementation based on the IBuySpy Portal Solution Kit was released today. The code allows you to run multiple portals from a single codebase/database (portals are identified by their unique domain names). Effectively this allows you to run multiple independent websites from a single ASP.NET/SQL Server hosted account.</p> <p>The code base incorporates a wide variety of custom software enhancements implemented over the past year. Although too numerous to mention, these include many back-end "plumbing" enhancements to improve the overall design and object-oriented nature of the IBS reference implementation.</p> <p>In order to gain access to the source code (VB.NET only at this point), you simply need to Register on the following site:</p> <p>IBuySpy Workshop</p> <p>This is my way of giving back to the community...</p> <p>Merry Christmas & Happy New Year!</p> <hr/> <p>Shaun Walker Perpetual Motion Interactive Systems Inc. http://www.perpetualmotion.ca</p> <p>post reply edit post</p> <p>Moderate Post: 111390 [Delete Post Edit Post Moderation History Move Post]</p>

Figure 1.3

IBuySpy Workshop

The public release of the IBuySpy Workshop (see [Figure 1.4](#)) created such a surge in forum activity that it was all I could do to keep up with the feedback, especially because this all occurred during the Christmas holidays. I had a family vacation booked for the first two weeks of January, and I left for Mexico on January 2, 2003 (one week after the initial IBuySpy Workshop release). At the time, the timing of this family vacation seemed poor because the groundswell of interest in the IBuySpy Workshop seemed like it could really use my dedicated focus. However, in hindsight the timing could not have been better because it proved that the community could support itself—a critical element in any open source project. When I returned home from vacation, I was amazed at the massive response the release achieved. The IBuySpy Portal Forum became dominated with posts about the IBuySpy Workshop, and my Inbox was full of messages thanking me for my efforts and requesting me to provide support and enhancements. This certainly validated my decision to release the application as an open source project but also emphasized the fact that I had started a locomotive down the tracks and it was going to take some significant engineering to keep it on the rails.

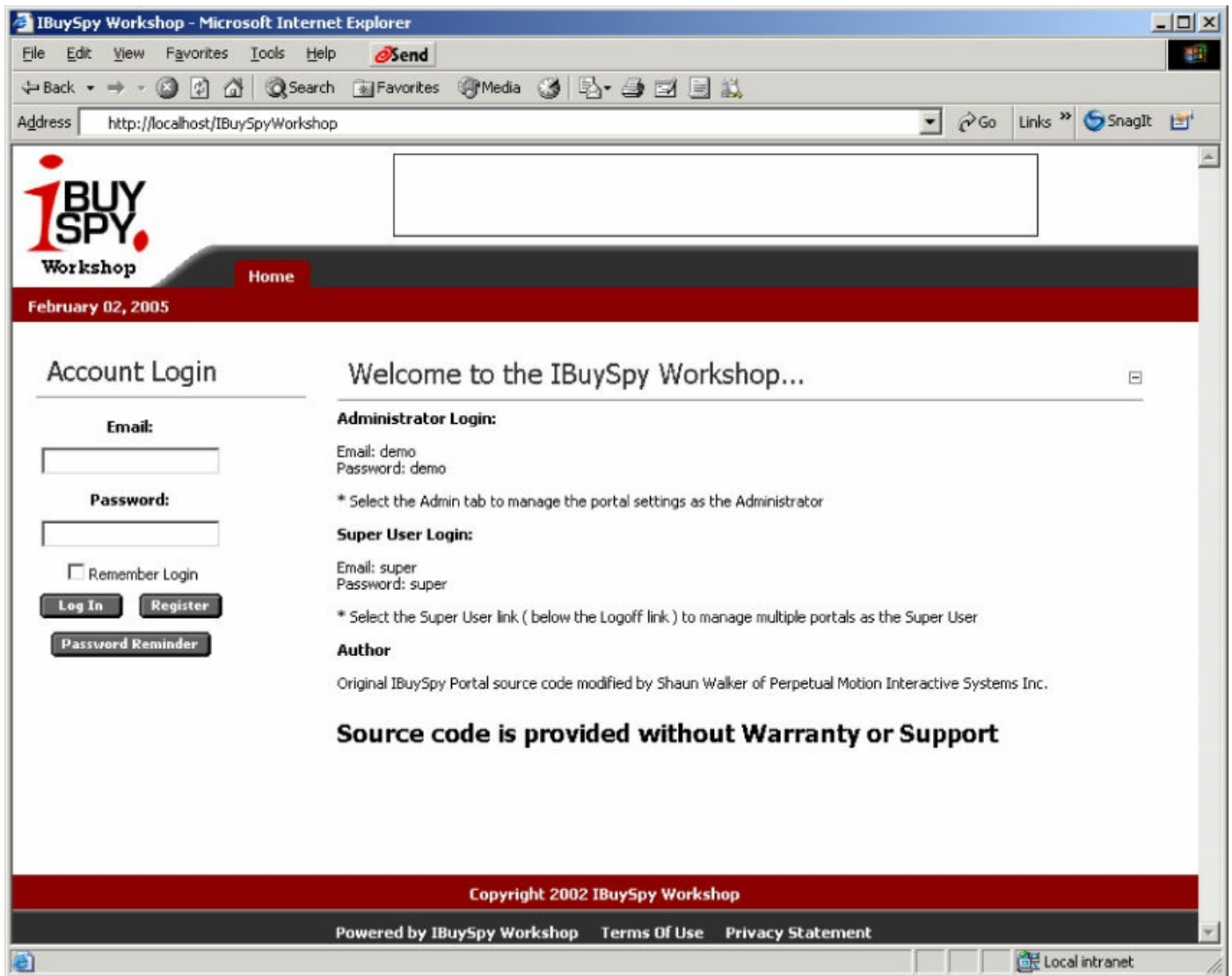


Figure 1.4

Over the next few months, I frantically attempted to incorporate all community suggestions into the application while at the same time keep up with the plethora of community support questions. Because I was working a day job that prevented effort on the open source project, most of my evenings were consumed with work on the IBuySpy Workshop, which definitely caused some strain on my marriage and family life. Four hours of sleep per night is not conducive to a healthy lifestyle, but, like I said, the train was rolling, and I had a feeling the project was destined for bigger things.

Supporting a user base through upgrades is fundamental in any software product. This is especially true in open source projects where the application can evolve quickly based on community feedback and technical advancements. The popular software notion is that “no user should be left on an evolutionary dead end.” As luck would have it, I had designed a reliable

upgrade mechanism in the original sports management application that I included in the IBuySpy Workshop code base. This feature enabled users of the application to easily migrate from one release version to the next—a critical factor in keeping the community engaged and committed to the evolution of the product.

In February 2003, the IBuySpy Portal Forum had become so congested with IBuySpy Workshop threads that it started to become difficult for the two communities to coexist peacefully. At this point, I sent an email to the webmaster address posted at the bottom of the forums page on the www.asp.net site with a request to create a dedicated forum for the IBuySpy Workshop. Because the product functionality and source code of the two applications diverged so significantly, my intent was to try to keep the forum posts for the two applications separated, providing both communities the means to support their membership. I certainly did not have high hopes that my email request was even going to be read—let alone granted. But to my surprise, I received a positive response from none other than Rob Howard (an ASP.NET icon), which proved to be a great introduction to a long-term partnership with Microsoft. Rob created the forum and even went a step further and added a link to the Source Download page of the www.asp.net site, an event that would ultimately drive a huge amount of traffic to the emerging IBuySpy Workshop community.

There are a number of reasons why the IBuySpy Workshop became so immediately popular when it was released in early 2003. The obvious reason is because the base application contained a huge number of enhancements over the IBuySpy Portal application, and people could immediately leverage them to build more powerful websites. From a community perspective, the open source project provided a central management authority that was dedicated to the ongoing growth and support of the application framework, a factor that was definitely lacking in the original IBuySpy Portal community. This concept of open source on the Microsoft platform attracted many developers—some with pure philosophical intentions, and others who viewed the application as a vehicle to further their own revenue-generating interests. Yet another factor, which I think is often overlooked, relates to the programming language on which the project was based. With the release of the .NET Framework 1.0, Microsoft spent a lot of energy promoting the benefits of the new C# programming language. The C# language was intended to provide a migration path for C++ developers as well as a means to

entice Java developers working on other platforms to switch. This left the Visual Basic and ASP 3.0 developer communities feeling neglected and somewhat unappreciated. The IBuySpy Workshop, with its core framework in VB.NET, provided an essential community ecosystem where legacy VB developers could interact, learn, and share.

Subscription Fiasco

In late February 2003, the lack of sleep, family priorities, and community demands finally came to a head and I decided that I should reach out for help. I contacted a former employer and mentor, Kent Alstad, with my dilemma, and we spent a few lengthy telephone calls brainstorming possible outcomes. However, my personal stress level at the time and my urgency to change direction on the project ultimately caused me to move too fast and with more impulsiveness than I should have. I announced that the IBuySpy Workshop would immediately become a subscription service where developers would need to pay a monthly fee to get access to the latest source code. From a personal perspective, the intent was to generate enough revenue that I could leave my day job and focus my full energy on the management of the open source project. And with 2,000 registered users, a subscription service seemed like a viable model (see [Figure 1.5](#)).

Author	Thread: Subscription Model for IBSW
<ul style="list-style-type: none"> ● sbwalker 	<p>Subscription Model for IBSW Posted: 02-25-2003 12:49 AM</p> <p>Dear User,</p> <p>With the latest release, IBuySpy Workshop has undergone some serious changes in terms of its revenue model. Having evolved as a free developer-oriented site, the IBSW has outgrown its limited resources and simply can not continue without some type of formalized cash flow. As a result, IBSW is being migrated to a subscription based model where users must pay a monthly fee to gain access to the product/services. Please note this change does not mean the project is dropping its Open Source philosophy, as the full source version will still be available for download.</p> <p>With careful consideration the current users of the IBSW have been categorized into two distinct groups. The Standard group has downloaded the application and is using its features to operate Internet/Intranet websites (which may include leveraging the multi-portal capability to sell subhosting services). The Developer group is more interested in examining/modifying the source code with the purpose of gaining insight/education into the new .NET platform. Although both of these groups have different perspectives, they both feel there is significant value in the current product and have a desire to be active in the emerging community.</p> <p>Having researched the current market, there is significant demand for an entry level portal/CMS application to cater to small to medium sized clients. The IBuySpy Workshop is well positioned to satisfy this market niche so long as the current momentum continues. And when you take into consideration the price tag for some of the commercial products in this area (ie. SharePoint = \$30,000, MS Content Management Server = \$50,000) the proposed subscription fee seems reasonable.</p> <p>Standard Membership = \$19.95/month (full-featured application does not include source code) Developer Membership = \$29.95/month (full-featured application including source code) Lifetime Membership (contributing members receive free access to all resources)</p> <p>There are already those in the community who have created Custom Modules for IBSW and are in the process of offering them for sale. It is not surprising that there is a significant opportunity for developers to generate revenue from their efforts. We are currently working on addressing some of the architectural limitations of the original IBS Portal so that Custom Modules can be offered as seamless plug-ins (without dealing with recompilation or database scripting issues). IBSW is a managed code base and will not be limited by the static problems imposed by the original IBS Portal.</p> <p>I am sure the change in revenue model will likely alienate some developers who feel the intentions of IBSW have been misrepresented in some way. But the fact is commercialization is sometimes a necessary evil in order to be able to achieve higher goals. As a result of the changes, IBSW will remain community funded, community focussed, and community driven.</p> <p>Thank you, we appreciate your support...</p> <p>IBuySpy Workshop</p> <hr/> <p>Shaun Walker Perpetual Motion Interactive Systems Inc. http://www.perpetualmotion.ca</p> <p>post reply edit post</p> <p style="text-align: right;">Moderate Post: 155889 [Delete Post Edit Post Moderation History Move Post]</p>

Figure 1.5

However, the true philosophy of the open source model immediately came to light, and I had to face the wrath of a scorned community. Among other things, I was accused of misleading the community, lying about the open source nature of the project, and letting my personal greed cloud my vision.

For every one supporter of my decision, there were 10 more who publicly crucified me as the evil incarnate. Luckily for me, Kent had a trusted work associate named Andy Baron, a senior consultant at MCW Technologies and a Microsoft Most Valuable Professional since 1995, who has incredible wisdom when it comes to the Microsoft development community. Andy helped me craft a public apology message (see [Figure 1.6](#)) that managed to appease the community and restore the IBuySpy Workshop to full open source status.

The image shows a screenshot of a forum post. At the top right, there are links for 'Previous Thread' and 'Next Thread'. The post header includes the author 'sbwalker' and the thread title 'And now the good news...'. The post content is a public apology message. At the bottom of the post, there are buttons for 'post reply' and 'edit post'. Below the post, there is a moderation bar with the text 'Moderate Post: 157019' and links for 'Delete Post', 'Edit Post', 'Moderation History', and 'Move Post'.

Author: sbwalker

Thread: And now the good news...

And now the good news...
Posted: 02-25-2003 10:52 PM

Well, I guess I've done a pretty good job of proving that I'm a much worse businessman than developer.

I'm thankful to the community that the overwhelming tone of your response has been supportive and forgiving. But I realize now that I made a big mistake by proposing a subscription model. It wasn't fair, and it probably wouldn't have worked anyway.

I wish I could just roll back time, but I'm doing the next best thing: I humbly withdraw the proposed change to a subscription model. If you paid for a subscription, I'll gladly refund your money. In the future, if you wish to contribute financially, I will implement a donations option on the site.

I'm considering all the suggestions that came in, and maybe I'll implement some of them. But the only suggestions I'm considering are things like providing a way to make voluntary contributions, or offering optional modules or services for sale.

I've completely abandoned any thought of demanding payment from those who want to continue participating in this great community project.

I'm also pleased to report that Microsoft has offered to kick in to support the continued growth of the IBSW community. We haven't worked out the details, and I'm not even sure it'll come to anything, but I've seen that they want to keep this open process going, and I'm committed to making that happen.

So, please forgive me, and let's get back to talking about code not money. I promise not to screw up like that again!

And last, but not least, the latest Open Source release (version 1.0.4) is now available on the site for download by Registered Users.

Shaun Walker
Perpetual Motion Interactive Systems Inc.
<http://www.perpetualmotion.ca>

[post reply](#) [edit post](#)

Moderate Post: 157019 [[Delete Post](#) | [Edit Post](#) | [Moderation History](#) | [Move Post](#)]

Figure 1.6

Microsoft

Coincidentally, the political nightmare I created in the IBuySpy Workshop Forum with my subscription announcement resulted in some direct attention from the Microsoft ASP.NET product team (the maintainers of the www.asp.net site). Still trying to recover from the damage I incurred to the goodwill of the project, I received an email from none other than Scott Guthrie (co-founder of the Microsoft ASP.NET Team), asking me to reexamine my decision on the subscription model and making suggestions on how the project could continue as a free, open source venture. It seemed that Microsoft was protective of its evolving community and did not want to see the progress in this area splinter and dissolve just as it seemed to be gaining momentum. Scott Guthrie made no promises at this point, but he did open a direct dialogue that ultimately led to some fundamental discussions on sponsorship and collaboration. In fact, this initial email led to a number of telephone conversations and ultimately an invitation to Redmond to discuss the future of the IBuySpy Workshop.

I still remember the combination of nerves and excitement as I drove from my home in Abbotsford, British Columbia, to Microsoft's head office in Redmond, Washington (about a 3-hour trek). I really did not know what to expect, and I tried to strategize all possible angles. Essentially all of my planning turned out to be moot because my meeting with Scott Guthrie turned out to be far more laidback and transparent than I could have ever imagined. Scott took me to his unassuming office, and we spent the next 3 hours brainstorming ideas about how the IBuySpy Workshop fit into the current ASP.NET landscape. Much of this centered on the evolving vision of ASP.NET 2.0—an area where I had little or no knowledge prior to the meeting (the Whidbey Alpha had not even been released at this point).

At the beginning of the meeting, Scott had me demonstrate the current version of the IBuySpy Workshop, explaining its key features and benefits. We also discussed the long-term goals of the project as well as my proposed roadmap for future enhancements. Scott's knowledge of both the technical and community aspects of the ASP.NET platform really amazed me—I guess that's why he is the undisputed “Father of ASP.NET.” In hindsight, I can hardly believe my good fortune to have received three dedicated hours of his time to discuss the project—it really changed my “ivory tower” perception of Microsoft and forged a strong relationship for future collaboration.

Upon leaving Redmond, I had to stifle my excitement as I realized that, regardless of the direct interaction with Microsoft, I personally was still in the same situation as before the subscription model announcement. Because the subscription model failed to generate the much-needed revenue that would have allowed me to devote 100 percent of my time to the project, I was forced to examine other possible alternatives. There were a number of suggestions from the community and the concept that seemed to have the most potential was related to web hosting.

In these early stages, there were few economical Microsoft Windows hosting options available that offered an SQL Server database—a fundamental requirement for running the IBuySpy Workshop application. Coincidentally, I had recently struck up a relationship with an individual from New Jersey who was active in the IBuySpy Workshop forums on www.asp.net. This individual had a solid background in web hosting and proposed a partnership whereby he would manage the web hosting infrastructure, and I would continue to enhance the application and drive traffic to the business. Initially there were a lot of community members who signed up for this service—some because of the low-cost hosting option, others because they were looking for a way to support the open source project. It soon became obvious that the costs to build and support the infrastructure were consuming the majority of the revenue generated. And over time the amount of effort to support the growing client base became more intense. Eventually it came to a point where it was intimated that my contributions to the web hosting business were not substantial enough to justify the current partnership structure. I was informed that the partnership should be dissolved. This is where things got complicated because there was never any formal agreement signed by either party to initiate the partnership. Without documentation, it made the negotiation for a fair settlement difficult and resulted in some bad feelings on both sides. This was unfortunate because I think the relationship was formed with the best intentions, but the demands of the business resulted in a poor outcome. Regardless, this ordeal was an important lesson I needed to learn: regardless of the open source nature of the project, it was imperative to have all contractually binding items properly documented.

DotNetNuke

One of the topics that Scott Guthrie and I discussed in our early conversations was the issue of product branding. IBuySpy Workshop achieved its early goals of providing a public reference to the IBuySpy Portal community. This resulted in an influx of ASP.NET developers who were familiar with the IBuySpy Portal application and were interested in this new open source concept. But as the code bases diverged, there was a need for a new project identity—a unique brand that would differentiate the community and provide the mechanism for building an internationally recognized ecosystem. Research of competing portal applications on other platforms revealed a strong tendency toward the “nuke” slogan.

The “nuke” slogan was originally coined by Francisco Burzi of PHP-Nuke fame (a pioneering open source content management system). Over the years, a variety of other projects adopted the slogan as well—so many that the term had obtained industry recognition in the portal-application genre. To my surprise, a WHOIS search revealed that dotnetnuke.com, [.net](http://dotnetnuke.net), and [.org](http://dotnetnuke.org) were not registered and, in my opinion, seemed to be the perfect identity for the project. Again emphasizing the bare-bones resources under which the project was initiated, my credit card transaction to register the three domain names was denied, and I was only able to register dotnetnuke.com (in the long run an embarrassing and contentious issue as the [.net](http://dotnetnuke.net) and [.org](http://dotnetnuke.org) domain names were immediately registered by other individuals). Equally as spontaneous, I did an Internet search for images containing the word “nuke” and located a three-dimensional graphic of a circular gear with a nuclear symbol embossed on it. I contacted the owner of the site and was given permission to use the image (it was, in fact, simply one of many public domain images they were using for a fictitious storefront demonstration). A new project identity was born—version 1.0.5 of the IBuySpy Workshop was rebranded as DotNetNuke, which the community preferred to abbreviate to “DNN” for simplicity (see [Figure 1.7](#)).

Author	Thread: Community Re-branding...
<p>● sbwalker</p>	<p>Community Re-branding... Posted: 03-02-2003 10:16 AM</p> <p>IBuySpy is an industry recognized brand for the Microsoft platform and has proven to be a good marketing tool for getting the the new Workshop open source community off the ground. However, moving forward I am proposing we drop the IBuySpy branding to differentiate our community from the other reference implementations (IBuySpy, Starter Kits).</p> <p>I recently purchased the www.dotnetnuke.com domain. The Nuke brand has significant industry recognition in the PHP/MySQL/Linux open source community in regards to Portal/Content Management Systems. With derivatives such as PHP-Nuke, MyPHPNuke, and PostNuke it has achieved considerable success during its short lifetime. Market competitors such as Java are also planning on leveraging this brand to promote their platform (www.javanuke.com). I think this is a great opportunity for our community (both IBSW and Windows) to distinguish ourselves.</p> <p>Here is a preliminary look at a conceptual logo:</p> <p>DotNetNuke</p> <p>What are your thoughts??</p> <hr/> <p>Shaun Walker Perpetual Motion Interactive Systems Inc. http://www.perpetualmotion.ca</p> <p>post reply edit post</p> <p>Moderate Post: 160986 [Delete Post Edit Post Moderation History Move Post]</p>

Figure 1.7

Licensing

A secondary issue that was not addressed during the early stages of the project was licensing. The original IBuySpy Portal was released under a liberal Microsoft EULA license that allowed for unrestricted usage, modification, and distribution. However, the code base underwent such a major transformation that it could hardly be compared with its predecessor. Therefore, when the IBuySpy Workshop application was released, I did not include the original Microsoft EULA, nor did I include any copyright or license of my own. Essentially this meant that the application was in the public domain. This is certainly not the most accepted approach to an open source project, and eventually some of the more legal-savvy community members brought the issue to a head. I was forced to take a hard look at open source licensing models to determine which license was most appropriate for the project.

In stark contrast to the spontaneous approach taken to finding a project identity, the licensing issue had much deeper ramifications. Had I not performed extensive research on this subject, I would have likely chosen a GPL license because it seemed to dominate the vast majority of open source projects in existence. However, digging beneath the surface, I quickly realized that the GPL did not seem to be a good candidate for my objectives of allowing DotNetNuke to be used in both commercial and noncommercial environments. Ultimately, the selection of a license for an open source project is largely dependent upon your business model, your product architecture, and understanding who owns the intellectual property in your application. The combination of these factors prompted me to take a hard look at the open source licensing options available.

For those of you who have not researched open source software, you would be surprised at the major differences between the most popular open source licensing models. It is true that these licenses all meet the standards of the Open Source Definition, a set of guidelines managed by the Open Source Initiative (OSI) at opensource.org. These principles include the right to use open source software for any purpose, the right to make and distribute copies, the right to create and distribute derivative works, the right to access and use source code, and the right to combine open source and other software. With such fundamental rights shared among all open source licenses, it probably makes you wonder why there is need for more than one license at all. Well, the reason is because each license has the ability to impose additional rights

or restrictions on top of these base principles. The additional rights and restrictions have the effect of altering the license so that it meets the specific objectives of each project. Because it is generally bad practice to create brand-new licenses (based on the fact that the existing licenses have gained industry acceptance as well as a proven track record), people generally gravitate toward either a GPL or BSD license.

The GPL License (or GNU General Public License) was created in 1989 by Richard Stallman, founder of the Free Software Foundation. The GPL License is what is now known as a “copyleft” license, a term coined based on its controversial reciprocity clause. Essentially, this clause stipulates that you are allowed to use the software on the condition that any derivative works that you create from it and distribute must be licensed to all under the same license. This is intended to ensure that the software and any enhancements to it remain in the public domain for everyone to share. Although this is a great humanitarian goal, it seriously restricts the use of the software in a commercial environment.

The MIT License (a variation of the Berkeley Software Distribution License) was created by the University of California and was designed to permit the free use, modification, and distribution of software without any return obligation on the part of the community. The MIT License is essentially a “copyright” license, meaning that you are free to use the software on the condition that you retain the copyright notice in all copies or derivative works. The MIT License is also known as an “academic” license because it provides the highest degree of intellectual property sharing.

Ultimately, I settled on a standard MIT License for DotNetNuke—a license that allows the maximum licensing freedom in both commercial and noncommercial environments with only minimal restrictions to preserve the copyright of the project. The change in license went widely unnoticed by the community because it did not impose any additional restrictions on usage or distribution. However, it was a fundamental milestone in establishing DotNetNuke as a true open source project:

```
DotNetNuke(r) - http://www.dotnetnuke.com  
Copyright (c) 2002-2003 by Perpetual Motion Interactive Systems Inc.  
(http://www.perpetualmotion.ca)  
Permission is hereby granted, free of charge, to any person obtaining  
a copy of  
this software and associated documentation files (the "Software"), to  
deal in
```

the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Core Team

The next major milestone in the project's open source evolution occurred in the summer of 2003. Up until this point, I had been acting as the sole maintainer of the DotNetNuke code base, a task that was consuming 110 percent of my free time as I feverishly fixed bugs and enhanced the framework based on community feedback. Still, I felt more like a bottleneck than a provider in spite of the fact that I was churning out at least one significant release every month leading up to this point. The more active community members were becoming restless due to a lack of direct input into the progress of the project. In fact, a small faction of these members even went so far as to create their own hybrid or “fork” of the DotNetNuke code base that attempted to forge ahead and add features at a more aggressive pace than I was capable of on my own. These were challenging times from a political standpoint because I was eventually forced to confront all of these issues in a direct and public manner—flexing my “benevolent dictator” muscles for the first time—an act I was not the least bit comfortable performing. Luckily for me, I had a number of loyal and trustworthy community members who supported my position and ultimately provided the backing to form a strong and committed Core Team.

Most successful open source projects are comprised of a number of community volunteers who earn their positions of trust and authority within the community based on their specific expertise or community support activities. This is known as a *meritocracy*, a term that means that an individual's influence is directly proportional to the ability that the individual demonstrates within the project. It's a well-observed fact that individuals with more experience and skills have less time to devote to volunteer activities; however, their minimal contributions prove to be incredibly valuable. Similarly, individuals with less experience may be able to invest more time but may only be capable of performing the more repetitive, menial tasks. Building a healthy balance of these two roles is exactly what is required in every successful open source project and, in fact, is one of the more challenging items to achieve from a management perspective.

The original DotNetNuke Core Team members were selected based on their participation and dedication to the DotNetNuke project in the months leading up to the team's formation. In most cases, this was solely based on an individual's public image and reputation established in the DotNetNuke

Forum on the www.asp.net website. And in fact, in these early stages, the online persona of each individual proved to be a good indicator of the specific skills he or she could bring to the project. Some members were highly skilled architects, others were seasoned developers, and others were better at discussing functionality from an end-user perspective and providing quality support to their community peers.

To establish some basic structure for the newly formed Core Team, I attempted to summarize some basic project guidelines. My initial efforts combined some of the best Extreme Programming (XP) rules with the principles of other successful open source projects. This became the basis of the DotNetNuke Manifest document:

- **Development is a team effort.** The whole is exponentially greater than the sum of its parts. Large-scale open source projects are viable only if a large enough community of highly skilled developers can be amassed to attack a problem. Treating your users as co-developers is your most effective option for rapid code improvement and effective debugging.
- **Build the right product before you build the product right.** Focus should be directed at understanding and implementing the high-level business requirements before attempting to construct the perfect technical architecture. Listen to your customers.
- **Incremental development.** Every software product has infinite growth potential if managed correctly. Functionality should be added in incremental units rather than attempting a monolithic implementation. Release often but with a level of quality that instills confidence.
- **Law of diminishing return.** The majority of the effort should be invested in implementing features that have the most benefit and widest general usage by the community.

DotNetNuke version 1.0.10 was the proving grounds for the original Core Team. The idea was to establish the infrastructure to support disconnected team development by working on a stabilization release of the current product. A lot of debate went into the selection of the appropriate source-control system because, ironically enough, many of the Core Team had never worked with a formal source control process in the past (a fact that certainly emphasized the varied professional background of the Core Team members). The debate centered on which source control system to use and ultimately the familiarity with the Microsoft model won out. We decided to use the free

WorkSpaces service on the GotDotNet website (a developer community site supported by Microsoft). GotDotNet also provided a simplistic Bug Tracker application that provided us with a means to manage the tracking of issues and enhancement requests. With these infrastructure components in place, we were able to focus on the stabilization of the application, correcting known defects and adding some minor usability enhancements. It was during this time that Scott Willhite stepped forward to assume a greater role of responsibility in the project: assisting in management activities, communication, prioritization, and scheduling.

A significant enhancement that was introduced in this stabilization release came from a third party who had contacted me with some specific enhancements they had implemented and wanted to contribute. The University of Texas at El Paso had done extensive work making the DotNetNuke application compliant with the guidelines of the American Disabilities Association (ADA) and Section 508 of the United States Rehabilitation Act. The U.S. government made compliancy mandatory for most public organizations; therefore, this was a great enhancement for DotNetNuke because it allowed the application to be used in government, educational, and military scenarios. Bruce Hopkins became the Core Team owner of this item in these early stages, a role that required a great deal of patience as the rest of the team came to grips with the new concept.

Establishing and managing a team was no small challenge. On one hand, there were the technical challenges of allowing multiple team members, all in different geographic regions, to communicate and collaborate in a cost-effective, secure environment. Certainly this would have never been possible without the Internet and its vast array of online tools. On the other hand, there was the challenge of identifying different personality types and channeling them into areas where they would be most effective. Because there are limited financial motivators in the open source model, people must rely on more basic incentives to justify their volunteer efforts. Generally this leads to a paradigm where contributions to the project become the de facto channel for building a reputation within the community—a primary motivator in any meritocracy. As a result of working with the team, it soon became obvious that there were two extremes in this area: those who would selflessly sacrifice all of their free time (often to their own detriment) to the open source project, and those who would invest the minimal effort and expect the maximum reward. As the creator and maintainer of the project it was my duty

to remain objective and put the interests of the community first. This often caused me to become frustrated with the behavior of specific individuals, but in nearly all cases these issues could be resolved without introducing any hard feelings on either side. This is true in all cases except one.

XXL Fork

Early in the project history, I was approached by an individual from Germany with a request to maintain a localized DotNetNuke site for the German community. I was certainly not naïve to the risks of another source code distribution at this point, and I told him that it would be fine so long as the site stayed consistent with the official source code base, which was under my jurisdiction. This was agreed upon, and in the coming months I had periodic communication with this individual regarding his localization efforts.

However, as time wore on he became critical of the manner in which the project was being managed, in particular the sole maintainer aspect, and began to voice his disapproval in the public forum. There was a group who believed that there should be a greater degree of transparency in the project—that developers should be able to get access to the latest development source code at any time and that the maintenance of the application should be conducted by a team rather than an individual. He was able to convince a number of community members to collaborate with him on a modified version of DotNetNuke, a version that integrated a number of the more popular community enhancements available, and called it DotNetNuke XXL.

Now I have to admit that much of this occurred due to my own inability to respond quickly and form a Core Team. In addition, I was not providing adequate feedback to the community regarding my goals and objectives for the future of the project. The reality is that the background management tasks of creating the DotNetNuke Core Team and managing the myriad other issues had undermined my ability to deliver source code enhancements and support to the community. The combination of these factors resulted in an unpleasant situation, one that I should have mitigated sooner but was afraid to act upon due to the fragility of the newly formed community. And you also need to remember that the creator of the XXL variant had broken no license agreement by creating a “fork” (a distribution)—it was completely legal based on the freedom of the MIT open source license.

Eventually the issue came to a head when members of the XXL group began promoting their full-source-code hybrid in the DotNetNuke Forum. Essentially piggybacking on the primary promotion channel for the DotNetNuke project, they were able to convince many people to switch to the XXL code base. This had some bad consequences for the DotNetNuke community. Mainly it threatened to splinter the emerging community on

territorial boundaries—an event I wanted to avoid at all costs. This situation was the closest attempt of project hijacking that I can realistically imagine. The DotNetNuke XXL fork seemed to be fixated on becoming the official version of DotNetNuke and assuming control of the future project roadmap. The only saving grace was that I personally managed the DotNetNuke infrastructure and therefore had some influence over key aspects of the open source environment.

In searching for an effective mechanism to protect the integrity of the community and prevent the XXL fork from gaining momentum, some basic business fundamentals came into play. Any product or service is only as successful as its promotion or marketing channel. The DotNetNuke Forum on the www.asp.net website was the primary communication hub to the DotNetNuke community. Therefore, it was not difficult to realize that restricting discussion on XXL in the forum was the simplest method to mitigate its growth. In probably the most aggressive political move I have ever been forced to make, I introduced some bold changes to the DotNetNuke project. I established some guidelines for Core Team conduct that included, among other things, restrictions on promoting competing open source distributions of the DotNetNuke application. I also posted some policies on the DotNetNuke Forum that emphasized that the forum was dedicated solely to the discussion of the official DotNetNuke application and that discussion of third-party commercial or open source products was strictly forbidden. This was an especially difficult decision to make from a moral standpoint as I was well aware that the DotNetNuke application had been introduced to the community via the IBuySpy Portal Forum. Nonetheless, the combination of these two announcements resulted in both the resignation of the XXL project leader from the Core Team as well as the end of discussion of the XXL fork in the DotNetNuke Forum. It is important to note that such a defensive move would not have been possible without the loyalty and support of the rest of the Core Team in terms of enforcing the guidelines.

The unfortunate side effect, one about which I had been cautioning members of the community for weeks, was that users who had upgraded to the XXL fork were effectively left on an evolutionary dead end—a product version with no support mechanism or promise of future upgrades. This is because many of the XXL enhancements were never going to be integrated into the official DotNetNuke code base (either due to design limitations or inapplicability to the general public). This situation, as unpleasant as it may have been for

those caught on the dead-end side of the equation, was a real educational experience for the community in general as they began to understand the longer-term and deeper implications of open source project philosophy. In general, the community feedback was positive to the project changes, with only occasional flare-ups in the weeks following. In addition, the Core Team seemed to gel more as a result of these decisions because it provided some much-needed policies on conduct, loyalty, and dedication as well as a concrete example of how inappropriate behavior would be penalized.

Trademarks

Emerging from the XXL dilemma, I realized that I needed to establish some legal protection for the long-term preservation of the project. Because standard copyright and the MIT license offered no real insurance from third-party threats, I began to explore intellectual property law in greater detail. After much research and legal advice, I decided that the best option was to apply for a trademark for the DotNetNuke name. Registering a trademark protects a project's name or logo, which is often a project's most valuable asset. After the trademark was approved, it would mean that although an individual or company could still create a fork of the application, they legally could not refer to it by the DotNetNuke name. This appeared to be an important distinction, so I proceeded with trademark registration in Canada (because this was the country in which Perpetual Motion Interactive Systems Inc. was incorporated).

I must admit the entire trademark approval process was quite an educational experience. Before you can register your trademark, you need to define a category and description of your wares and/or services. This can be challenging, although most trademark agencies now provide public access to their database where you can browse for similar items that have been approved in the past. You pay your processing fee when you submit the initial application, but the trademark agency has the right to reject your application for any number of reasons whereby you need to modify your application and submit it again. Each iteration can take a couple of months, so patience is indeed a requirement. After the trademark is accepted, it must be published in a public trademark journal for a specified amount of time, providing third parties the opportunity to contest the trademark before it is approved. If it makes it through this final stage, you can pay your registration fee for the trademark to become official. To emphasize the lengthy process involved, the DotNetNuke trademark was initially submitted on October 9, 2003, and was finally approved on November 15, 2004.

Sponsorship

In August 2003, I came to an agreement with Microsoft regarding a sponsorship proposal for the DotNetNuke project. In a nutshell, Microsoft wanted DotNetNuke to be enhanced in a number of key areas with the intent being to use the open source project as a means of demonstrating the strengths of the ASP.NET platform. Because these enhancements were completely congruent with the future goals of the project, there was little negative consequence from a technical perspective. In return for implementing the enhancements, Microsoft would provide a number of sponsorship benefits to the project including web hosting for the www.dotnetnuke.com website, weekly meetings with an ASP.NET Team representative (Rob Howard), continued promotion via the www.asp.net website, and more direct access to Microsoft resources for mentoring and guidance. It took five months for this sponsorship proposal to come together, which demonstrates the patience and perseverance required to collaborate with such an influential partner as Microsoft. Nonetheless, this was potentially a one-time offer, and at such a critical stage in the project evolution it seemed too important to ignore.

An interesting perception that most people have in the IT industry is that Microsoft is morally against the entire open source phenomenon. In my opinion, this is far from the truth—the reality is so much more simplistic. Like any other business that is trying to enhance its market position, Microsoft is merely concerned about competition. This is nothing new. In the past, Microsoft faced competitive challenges from many sources—companies, individuals, and governments. However, the environment at the time made it much more emotional and newsworthy to suggest that Microsoft was pitted against a grassroots community movement rather than a business or legal concern. And it took some time and effort for Microsoft to adapt to the changing landscape, but in recent years Microsoft has now embraced open source to remain competitive.

When it comes to DotNetNuke, many people probably questioned why Microsoft would be interested in assisting an open source project where it receives no direct benefit. And it may be perplexing why Microsoft would sponsor a product that competes to some degree with several of its own commercial applications. But you do not have to look much further than the obvious indirect benefits to see why this relationship has tremendous value.

First and foremost, the DotNetNuke application was only designed for use on the Microsoft platform. This meant that in order to use DotNetNuke, you needed to have valid licenses for a number of Microsoft infrastructure components (Windows operating system, database server, and so on). So this provided the financial value. In addition, DotNetNuke promoted the benefits of the .NET Framework and encouraged developers to migrate to Microsoft's development platform. This provides the educational value. Finally, it cultivated an active and passionate community—a network of loyal supporters who were motivated to leverage and promote Microsoft technology on an international scale. This provided the marketing value.

Enhancements

In September 2003, with the assistance of the newly formed Core Team, we embarked on an ambitious mission to implement the enhancements to DotNetNuke suggested by Microsoft. The problem at this point was that in addition to the Microsoft enhancements, there were some critical community enhancements, which I ultimately perceived as an even higher priority if the project should hope to grow to the next level. So the scope of the enhancement project began to snowball, and estimated release dates began to slip. The quality of the release code was also considered to be so crucial a factor that early beta packages were not deemed worthy of distribution. Ultimately, the code base evolved so much that there was little question the next release would need to be labeled version 2.0. During this phase of internal development, some members of the Core Team did an outstanding job of supporting the 1.x community and generating excitement about the next major release. This was critical in keeping the DotNetNuke community engaged and committed to the evolving project.

A number of excellent community enhancements for the DotNetNuke 1.0 platform also emerged during this stage. This sparked an active third-party reseller and support community, establishing yet another essential factor in any largely successful open source project. Unfortunately, at this point the underlying architecture of the DotNetNuke application was not particularly extensible, which made the third-party enhancements susceptible to upgrade complications and somewhat challenging to integrate for end users. As a Core Team, we recognized this limitation and focused on full modularity as a guiding principle for all future enhancements.

Modularity is an architecture principle that basically involves the creation of well-defined interfaces for the purpose of extending an application. The goal of any framework should be to provide interfaces in all areas that are likely to require customization based on business requirements or personalization based on individuality. DotNetNuke provides extensibility in the area of modules, skins, templates, data providers, and localization. And DotNetNuke typically goes one step beyond defining the basic interface: it actually provides the full spectrum of related resource services including creation, packaging, distribution, and installation. With all of these services available, it makes it extremely easy for developers to build and share application extensions with other members of the community.

One of the benefits of working on an open source project is the fact that there is a high priority placed on creating the optimal solution or architecture. This goal often results in more preliminary analysis and design that tends to elongate the schedule but also results in a more extensible and adaptable architecture. This differs from traditional application development that often suffers from time and budget constraints, resulting in shortcuts, poor design decisions, and delivery of functionality before it is validated. Another related benefit is that the developers of open source software also represent a portion of its overall user community, meaning they actually “eat their own dog food” so to speak. This is really critical when it comes to understanding the business requirements under which the application needs to operate. Far too often you find commercial vendors who build their software in a virtual vacuum, never experiencing the fundamental application use cases in a real-world environment.

One of the challenges in allowing the Core Team to work together on the DotNetNuke application was the lack of high-quality infrastructure tools. Probably the most fundamental elements from a software development standpoint were the need for a reliable source-code-control system and issue-management system. Because the project had little to no financial resources to draw upon, we were forced to use whatever free services were available in the open source community. And although some of these services are leveraged successfully by other open source projects, the performance, management, and disaster recovery aspects are sorely lacking. This led to a decision to approach some of the more successful commercial vendors in these areas with requests for pro-bono software licenses. Surprisingly, these vendors were more than happy to assist the DotNetNuke open source project in exchange for some minimal sponsorship recognition. This model has ultimately been carried on in other project areas to acquire the professional infrastructure, development tools, and services necessary to support our growing organization.

As we worked through the enhancements for the DotNetNuke 2.0 project, a number of Core Team members gained considerable respect within the project based on their high level of commitment, unselfish behavior, and expert development skills. Joe Brinkman, Dan Caron, Scott McCulloch, and Geert Veenstra sacrificed a lot of personal time and energy to improve the DotNetNuke open source project. And the important thing to realize is that they did so because they wanted to help others and make a difference, not

because of self-serving agendas or premeditated expectations. The satisfaction of working with other highly talented individuals in an open, collaborative environment is reward enough for some developers. And it is this particular aspect of open source development that continues to confound and amaze people as time goes on.

In October 2003, there was a Microsoft Professional Developers Conference (PDC) in Los Angeles, California. The PDC is the premier software development spectacle for the Microsoft platform; it's an event that occurs only every two years. About a month prior to the event Cory Isakson, a developer on the Rainbow Portal open source project (another derivative of the IBuySpy Portal based on C#), contacted me, saying that "Open Source Portals" had been nominated as a category for a "Birds of Feather" session at the event. I posted the details in the DotNetNuke Forum, and soon the item had collected enough community votes that it was approved as an official BOF session. This provided a great opportunity to meet with DotNetNuke enthusiasts and critics from all over the globe. It also provided a great networking opportunity to chat with the most influential commercial software vendors in the .NET development space (contacts made with SourceGear and MaximumASP at this event proved to be important to DotNetNuke, as time would tell).

Security Flaw

In January 2004, another interesting dilemma presented itself. I received an email from an external party, a web application security specialist who claimed to have discovered a severe vulnerability in the DotNetNuke application (version 1.0). Upon further research, I confirmed that the security hole was indeed valid and immediately called an emergency meeting of the more trusted Core Team members to determine the most appropriate course of action. At this point, we were fully focused on the DotNetNuke 2.0 development project but also realized that it was our responsibility to serve and protect the growing DotNetNuke 1.0 community. From a technical perspective, the patch for the vulnerability proved to be a simple code modification.

The more challenging problem was related to communicating the details of the security issue to the community. On the one hand we needed the community to understand the severity of the issue so that they would be motivated to patch their applications. On the other hand, we did not want to cause widespread alarm, which could lead to a public perception that DotNetNuke was an insecure platform. Exposing too many details of the vulnerability would be an open invitation for hackers to try to exploit DotNetNuke websites, but revealing too few details would downplay the severity. And the fact that the project is open source meant that the magnitude of the problem was amplified. Traditional software products have the benefit of tracking and identifying users through restrictive licensing policies. Open source projects have licenses that allow for free redistribution, which means the maintainer of the project has no way to track the actual usage of the application and no way to directly contact all community members who are affected.

The whole situation really put security issues into perspective for me. It's one thing to be an outsider, expressing your opinions on how a software vendor should or should not react to critical security issues in its products. It's quite another thing to be an insider, stuck in the vicious dilemma between divulging too much or too little information, knowing full well that both options have the potential to put your customers at even greater risk. Ultimately, we created a new release version and issued a general security alert that was sent directly to all registered users of the DotNetNuke application by email and posted in the DotNetNuke Forum on www.asp.net:

Subject: DotNetNuke Security Alert Yesterday we became aware of a security vulnerability in DotNetNuke. It is the immediate recommendation of the DotNetNuke Core Team that all users of DotNetNuke based systems download and install this security patch as soon as possible. As part of our standard security policy, no further detailed information regarding the nature of the exploit will be provided to the general public. This email provides the steps to immediately fix existing sites and mitigate the potential for a malicious attack. Who is vulnerable? -- Any version of DotNetNuke from version 1.0.6 to 1.0.10d What is the vulnerability? A malicious user can anonymously download files from the server. This is not the same download security issue that has been well documented in the past whereby an anonymous user can gain access to files in the /Portals directory if they know the exact URL. This particular exploit bypasses the file security mechanism of the IIS server completely and allows a malicious user to download files with protected mappings (ie. *.aspx). The vulnerability specifically *does not* enable the following actions: -- A hacker *cannot* take over the server (e.g. it does not allow hacker code to be executed on the server) How to fix the vulnerability? For Users: { Instructions on where to download the latest release and how to install } For Developers: { Instructions with actual source code snippets for developers who had diverged from the official DotNetNuke code base and were therefore unable to apply a general release patch } Please note that this public service announcement demonstrates the professional responsibility of the Core Team to treat all possible security exploits as serious and respond in a timely and decisive manner. We sincerely apologize for the inconvenience that this has caused. Thank you, we appreciate your support... DotNetNuke - The Web of the Future

The security dilemma brings to light another often misunderstood paradigm when it comes to open source projects. Most open source projects have a

license that explicitly states that there is no support or warranty of any kind for users of the application. And while this may be true from a purely legal standpoint, it does not mean that the maintainer of the open source application can ignore the needs of the community when issues arise. The fact is, if the maintainer did not accept responsibility for the application, the users would quickly lose trust and the community would dissolve. This implicit trust relationship is what all successful open source communities are based upon. So in reality, the open source license acts as little more than a waiver of direct liability for the maintainer. The DotNetNuke project certainly conforms to this model because we take on the responsibility to ensure that all users of the application are never left on an evolutionary dead end and security issues are always dealt with in a professional and expedient manner.

DotNetNuke 2.0

After six months of development, including a full month of public beta releases and community feedback, DotNetNuke 2.0 was released on March 23, 2004. This release was significant because it occurred at VS Live! in San Francisco, California, a large-scale software development event sponsored by Microsoft and Fawcette publications. Due to our strong working relationship with Microsoft, I was invited to attend official press briefings conducted by the ASP.NET Team. Essentially, this involved up to eight private sessions with the leading press agencies (Fawcette, PC Magazine, Computer Wire, Ziff Davis, and so on) where I was able to summarize the DotNetNuke project, show them a short demonstration, and answer their specific questions. The event proved to be spectacularly successful and resulted in a surge of new traffic to the community (now totaling more than 40,000 registered users).

DotNetNuke 2.0 was a hit. We had successfully delivered a high-quality release that encapsulated the majority of the most requested product enhancements from the community. And we had done so in a manner that allowed for clean customization and extensibility. In particular, the skinning solution in DotNetNuke 2.0 achieved widespread critical acclaim.

In DotNetNuke 1.X, the user interface of the application allowed for little personalization—essentially all DNN sites looked much the same, a negative restriction considering the highly creative environment of the World Wide Web. DotNetNuke 2.0 removed this restriction and opened the application to a whole new group of stakeholders: web designers. As the popularity of portal applications had increased in recent years, the ability for web designers to create rich, graphical user interfaces had diminished significantly. This is because the majority of portal applications were based on platforms that did not allow for clear separation between form and function or were architected by developers who had little understanding of the creative needs of web designers. DotNetNuke 2.0 focused on this problem and implemented a solution where the portal environment and creative design process could be developed independently and then combined to produce a stunningly harmonious end-user experience. The process was not complicated and did not require the use of custom tools or methodologies. It did not take long before we began to see DotNetNuke sites with richly creative and highly graphical layouts emerge, proving the effectiveness of the solution and creating a “Can you top this?” community mentality for innovative portal

designs.

[DotNetNuke.com](http://www.dotnetnuke.com) Website

To demonstrate the effectiveness of the skinning solution, I commissioned a local web design company, Circle Graphics in Abbotsford owned by Brad Haima, to create a compelling design for the www.dotnetnuke.com website (see [Figure 1.8](#)). As an open source project, I felt that I could get away with an unorthodox, somewhat futuristic site design, and I was impressed by some of Circle Graphics' futuristic, industrial concepts I had seen.



Figure 1.8

It turned out that the designer who had created these visuals, Anson Vogt, had since moved on but was willing to take on a small contract as a personal

favor to the owner. He created a skin that included some stunning 3-D imagery including the now infamous “nuke-gear” logo, circuit board, and plenty of twisted metallic pipes and containers. The integration with the application worked flawlessly, and the community was wildly impressed with the stunning result. Coincidentally, Anson Vogt later worked with musician Eminem as the Art Director for 3-D animation on the critically acclaimed Mosh video.

Provider Model

One of the large-scale enhancements that Microsoft insisted on for DotNetNuke 2.0 also proved to be popular. The Data Access Layer in DotNetNuke had been rearchitected using an abstract factory model that effectively allowed it to interface with any number of relational databases. Microsoft coined the term “provider model” and emphasized it as a key component in the future ASP.NET 2.0 framework. Therefore, getting a reference implementation of this pattern in use in ASP.NET 1.x had plenty of positive educational benefits for Microsoft and DotNetNuke developers. DotNetNuke 2.0 included both a fully functional SQL Server and Microsoft Access version, and the community soon stepped forward with MySQL and Oracle implementations as well. Again, the extensibility benefits of good architecture were extremely obvious and demonstrated the direction we planned to pursue in all future product development.

Upon review of the DotNetNuke 2.0 code base, it was obvious that the application bore little resemblance to the original IBuySpy Portal application. This was a good thing because it raised the bar significantly in terms of *n*-tiered, object-oriented, enterprise-level software development. However, it was also bad in some ways because it alienated some of the early DotNetNuke enthusiasts who were in fact “hobby programmers,” using the application more as a learning tool than a professional product. This is an interesting paradigm to observe in many open source projects. In the early stages, the developer community drives the feature set and extensibility requirements that, in turn, results in a much higher level of sophistication in terms of system architecture and design. However, as time goes on, this can sometimes result in the application surpassing the technical capabilities of some of its early adopters. DotNetNuke had ballooned from 15,000 lines of managed code to 46,000 lines of managed code in a little more than six months. The project was getting large enough that it required some serious effort to understand its organizational structure, dependencies, and development patterns.

Open Source Philosophy

When researching the open source phenomenon, there are a few fundamental details that are often ignored in favor of positive marketing rhetoric. I would like to take the opportunity to bring some of these to the surface because they provide some additional insight into some of the issues we face in the DotNetNuke project.

The first myth surrounds the belief that open source projects basically have an unlimited resource pool at their immediate disposal. Although this may be true from a purely theoretical perspective, the reality is that you still require a dedicated management structure to ensure that all of the resources are channeled in an efficient and productive manner. An army of developers without some type of central management authority will never consistently produce a cohesive application; and more likely, their efforts will result in total chaos. As much as the concept is often despised by hard-core programmers, dedicated management is absolutely necessary to set expectations and goals, ensure product quality, mitigate risk, recognize critical dependencies, manage scope, and assume ultimate responsibility. You will find no successful open source project that does not have an efficient and highly respected management team.

Also with regard to the unlimited resourcing myth, there are in fact few resources who become involved in an open source project that possess the level of competency and communication skills required to earn a highly trusted position in the meritocracy. More often, the resources who get involved are capable of handling more consumer-oriented tasks such as testing, support, and minor defect corrections. This is not to say that these resources do not play a critical role in the success of the project—every focused ounce of volunteer effort certainly helps sustain the health of the project. But my point is that there is usually a relatively small group on most open source projects who are responsible for the larger-scale architectural enhancements.

Yet another myth is related to the belief that anyone can make a direct and immediate impact on an open source project. Although this may be true to some degree, you generally need to build a trusted reputation within the community before you are granted any type of privilege. And there are few individuals who are ever awarded direct write access to the source code repository. Anyone has the ability to submit a patch or enhancement

suggestion; however, there's no guarantee that it will be added to the open source project code base. In fact, all submissions are rigorously peer-reviewed by trusted resources, and only when they have passed all validation criteria are they introduced to the source code repository. In addition, although a specific submission may appear to be quite useful when judged in isolation, there may be higher-level issues to consider in terms of upgrade support (a situation that can lead to submitter frustration if the issues are not fully explained). From a control standpoint, this is not much different than source control management on a traditional software project. However, the open source model does significantly alter this paradigm in that everyone is able to review the source code. As a result, the sheer volume of patches submitted to this process can be massive.

Stabilization

Following the success of DotNetNuke 2.0, we focused on improving the stability and quality of the application. Many production issues were discovered after the release that we would have never anticipated during internal testing. As an application becomes more extensible, people find ingenious new ways to apply it, which often produces unexpected results. We also integrated some key Roadmap enhancements that were developed in isolation by Core Team members. These enhancements were actually quite advanced because they added a whole new level of professional features to the DotNetNuke code base, transforming it into a viable enterprise application framework.

It was during this time that Dan Caron single-handedly made a significant impact on the project. Based on his experience with other enterprise applications, he proceeded to add integrated exception handling and event logging to DotNetNuke. This provided stability and “auditability”—two major factors in most professional software products. He also added a complex, multi-threaded scheduler to the application. The scheduler was not just a simple hard-coded implementation like I had seen in other ASP.NET projects, but rather it was fully configurable via an administrative user interface. This powerful new feature could be used to run background housekeeping jobs as well as long-running tasks. With this in place, the extensibility of the application improved yet again.

Third-Party Components

An interesting concern that came to our attention at this time was related to our dependence on external components. To provide the most powerful application, we had leveraged a number of rich third-party controls for their expert functionality. Because each of these controls was available under its own open source license, they seemed to be a good fit for the DotNetNuke project. But the fact is there are some major risks to consider. Some open source licenses are viral in nature and have the potential to alter the license of the application with which they are combined. In addition, there is nothing that prevents third parties from changing their licensing policy at any time. If this situation occurs, then it is possible that all users of the application who reference the control could be in violation of the new license terms. That's a fairly significant issue and certainly not something that can be taken lightly. Based on this knowledge, we quickly came up with a strategy that was aimed at minimizing our dependency on third-party components. We constructed a policy whereby we would always focus on building the functionality ourselves before considering an external control. And in the cases where a component was too elaborate to replicate, we would use a provider model, much like we had in the database layer, to abstract the application from the control in such a way that it would allow for a plug-in replacement. This strategy protects the community from external license changes and also provides some additional extensibility for the application.

With the great publicity on the www.asp.net website following VS Live! and the consistent release of powerful new enhancements, the spring of 2004 brought a lot of traffic to the dotnetnuke.com community website. At this point, the site was poorly organized and sparse on content due to a lack of dedicated effort. Patrick Santry had been on the Core Team since its inception, and his experience with building websites for the ASP.NET community became valuable at this time. We managed to make some fairly major changes to improve the site, but I soon realized that a dedicated resource would be required to accomplish all of our goals. Without the funding to secure such a resource, many of the plans had to unfortunately be shelved.

Core Team Reorganization

The summer of 2004 was a restructuring period for DotNetNuke. Thirty new community members were nominated for Core Team inclusion, and the Core Team itself underwent a reorganization of sorts. The team was divided into an Inner Team and an Outer Team. The Inner Team designation was reserved for those original Core Team individuals who had demonstrated the most loyalty, commitment, and value to the project over the past year. The Outer Team represented individuals who had earned recognition for their community efforts and were given the opportunity to work toward Inner Team status. Among other privileges, write access to the source code repository is the pinnacle of achievement in any source code project, and members of both teams were awarded this distinction to varying degrees.

In addition to the restructuring, a set of Core Team guidelines was established that helped formalize the expectations for team members. Prior to the creation of these guidelines, it was difficult to isolate nonperformers because there were no objective criteria by which they could be judged. In addition to the new recruits, a number of inactive members from the original team were retired, mostly to demonstrate that Core Team inclusion was a privilege, not a right. The restructuring process also brought to light several deficiencies in the management of intellectual property and confidentiality among team members. As a result, all team members were required to sign a retroactive nondisclosure agreement as well as an intellectual property contribution agreement. All of the items exemplified the fact that the project had graduated from its “hobby” roots to a professional open source project.

Microsoft Membership API

During these formative stages, I was once again approached by Microsoft with an opportunity to showcase some specific ASP.NET features. Specifically, a Membership API had been developed by Microsoft for Whidbey (ASP.NET 2.0), and it was planning on creating a backported version for ASP.NET 1.1 that we could leverage in DotNetNuke. This time the benefits were not so immediately obvious and required some thorough analysis. This is because DotNetNuke already had more functionality in these areas than the new Microsoft API could deliver. So to integrate the Microsoft components without losing any features, we would need to wrap the Microsoft API and augment it with our own business logic. Before embarking on such an invasive enhancement, we needed to understand the clear business benefit provided.

Well, you can never discount Microsoft's potential to impact the industry. Therefore, being one of the first to integrate and support the new Whidbey APIs would certainly be a positive move. In recent months there had been numerous community questions regarding the applicability of DotNetNuke with the early Whidbey Beta releases now in active circulation. Early integration of such a core component from Whidbey would surely appease this group of critics. From a technology perspective, the Microsoft industry had long been awaiting an API to converge upon in this particular area, making application interoperability possible and providing best practice due diligence in the area of user and security information. Integrating the Microsoft API would allow DotNetNuke to “play nicely” with other ASP.NET applications—a key factor in some of the larger-scale extensibility we were hoping to achieve. Last, but not least, it would further our positive relationship with Microsoft—a factor that was not lost on most as the key contributor to the DotNetNuke project's growth and success.

The reorganization of the Core Team also resulted in the formation of a small group of highly trusted project resources that, for lack of a better term, we named the Board of Directors. The members included myself, Scott Willhite, Dan Caron, Joe Brinkman, and Patrick Santry. The purpose of this group was to oversee the long-term strategic direction of the project. This included discussion on confidential issues pertaining to partners, competitors, and revenue. In August 2004, we scheduled our first general meeting for Philadelphia, Pennsylvania. With all members in attendance, we made some

excellent progress on defining action items for the coming months. This was also a great opportunity to finally meet in person some of the individuals with whom we had experienced only Internet contact in the past. With the first day of meetings behind us, the second day was dedicated to sightseeing in the historic city of Philadelphia. The parallels between the freedom symbolized by the Liberty Bell and the software freedom of open source were not lost on any of us that day.

Returning from Philadelphia, I knew that I had some significant deliverables on my plate. We began the Microsoft Membership API integration project with high expectations of completion within three months. But as before, there were a number of high-priority community enhancements that had been promised prior to the Microsoft initiative, and as a result the scope snowballed. Scope management is an extremely difficult task when you have such an active and vocal community.

“Breaking” Changes

The snowball effect soon revealed that the next major release would need to be labeled version 3.0. This is mostly because of “breaking” changes: modifications to the DotNetNuke core application that changed the primary interfaces to the point that plug-ins from the previous version 2.0 release would not integrate without at least some minimal changes. The catalyst for this was due to changes in the Membership API from Microsoft, but this only led to a decision of “If you are forced to break compatibility, introduce all of your breaking changes in one breaking release.” The fact is there was a lot of baggage preserved from the IBuySpy Portal that we were restricted from removing due to legacy support considerations. DotNetNuke 3.0 provided the opportunity to reexamine the entire project from a higher level and make some of the fundamental changes we had been delaying for years in some cases. This included the removal of a lot of dead code and deprecated methods as well as a full namespace reorganization that finally accurately broke the project API into logical components.

DotNetNuke 3.0 also demonstrated another technical concept that would both enrich the functionality of the application framework as well as improve the extensibility without the threat of breaking binary compatibility. Up until version 3.0, the service architecture for DotNetNuke was completely unidirectional. Custom modules could consume the resources and services offered by the core DotNetNuke framework but not vice versa. So although the application managed the secure presentation of custom modules within the portal environment, it could not get access to the custom module content information. Optional interfaces were added to enable custom modules to provide plug-in implementations for defined core portal functions. They also provided a simple mechanism for the core framework to call into third-party modules, providing a bidirectional communication channel so that modules could finally offer resources and services to the core.

Web Hosters

Along with its many technological advances, DotNetNuke 3.0 was also being groomed for use by entirely new stakeholders: web hosters. For a number of years, the popularity of Linux hosting has been growing at a far greater pace than Windows hosting. The instability arguments of early Microsoft web servers were beginning to lose their weight as Microsoft released more resilient and higher-quality server operating systems. Windows Server 2003 had finally shed its clunky Windows NT 4.0 roots and was a true force to be reckoned with. Aside from the obvious economic licensing reasons, there was another clear reason why hosters were still favoring Linux over Windows for their clients: the availability of end-user applications.

The Linux platform had long been blessed with a plethora of open source applications running on the Apache web server, built with languages such as PHP, Perl, and Python, and leveraging open source databases such as MySQL. (The combination of these technologies is commonly referred to as LAMP.) The Windows platform was really lacking in this area and was desperately in need of applications to fill this void.

For DotNetNuke to take advantage of this opportunity, it needed a usability overhaul to transform it from a niche developer-oriented framework to a polished end-user product. This included a usability enhancement from both the portal administration as well as the web host perspectives. Since Rob Howard left Microsoft in June 2004, my primary Microsoft contact was Shawn Nandi. Shawn did a great job of drawing upon his usability background at Microsoft to come up with suggestions to improve the DotNetNuke end-user experience. Portal administrators received a multilingual user interface with both field-level and module-level help. Enhanced management functions were added in key locations to improve the intuitive nature of the application. Web hosters received a customizable installation mechanism. In addition, the application underwent a security review to enable it to run in a Medium Trust—Code Access Security (CAS) environment. The end result was a powerful open source, web-application framework that could compete with the open source products on other platforms and offer web hosters a viable Windows alternative for their clients.

DotNetNuke 3.0

Much of the integration work on the Membership API and usability improvements were fueled by a much larger hosting initiative that Microsoft was preparing to unleash in May 2005. This initiative included a comprehensive program aimed at increasing awareness for Windows-based hosting solutions on an international level. Based on its strength as a framework for building consumer websites, Microsoft invited DotNetNuke to participate in the program as long as it could meet a defined set of technical criteria, including Membership API integration, Medium Trust CAS compliance, localization, and usability improvements. Nearly all of the enhancements were already identified on the product roadmap, so the opportunity to be included in the hosting program was really a win-win proposition for the project and the community. In addition, we believed that the benefit of participating in such a large-scale initiative would be enormous in terms of lending credibility to the DotNetNuke product, introducing the project to influential new stakeholders, and helping to build brand equity.

Core Team members made significant contributions during the development of DotNetNuke 3.0. Scott McCulloch, with the assistance of Jeremy White, implemented a full-featured URL rewriting component that allowed DotNetNuke to use standard URLs. Vicenç Masanas was instrumental in working on localization, templating, and stabilization tasks. Joe Brinkman implemented search-engine architecture, enabling content indexing across all modules in a portal instance. Jon Henning introduced a Client API library, enabling powerful client-side behavior in DotNetNuke modules. Perhaps the greatest code contributions were made by Charles Nurse. Realizing the massive amount of work that would be required to deliver the enhancements for the hosting program (and knowing that using only volunteer efforts would not hit the schedule deadlines), I hired the first full-time DotNetNuke contract resource. Charles was immediately put to work abstracting all of the core modules into independent private assemblies. At the same time, he reorganized entry fields in all application user interfaces and added full localization capabilities, including field-level online help.

The concept of localization was one of the most commonly requested enhancements for the DotNetNuke application. Localization actually has multiple meanings when it comes to software applications because there is a distinct difference between static and dynamic content. Static content is

information that is delivered as part of the core application typically implemented by developers. Dynamic content is information that is provided by users of the application and is typically entered by knowledge workers or webmasters. In DotNetNuke 3.0, we delivered full static localization for all administrative interfaces. This meant that all labels, messages, and help text could be translated and displayed in different languages based on the preference of the user. Developing a scalable architecture in this area turned out to be a challenging task because the solutions offered by Microsoft as part of the ASP.NET 1.x framework were better suited for desktop applications and had serious deficiencies and limitations for web applications. Instead, we decided to target the ASP.NET 2.0 localization architecture, which better addressed the web scenario. However, due to the specific business requirements of DotNetNuke, we soon realized that we were going to have to take some liberties with the proposed ASP.NET 2.0 localization architecture to enable us to achieve our goals for runtime updatability and scalability in a shared hosting environment. In the end, we were able to deliver a powerful solution that satisfied our business needs and provided forward compatibility to the upcoming ASP.NET 2.0 release.

The optional interface architectural model described earlier reaped rewards in DotNetNuke 3.0 in a number of key application areas. Registration of module actions in earlier versions of DotNetNuke was always less than optimal because they were dependent on page life-cycle events that were difficult to manage in a variety of scenarios. Optional interfaces finally provided a clean mechanism for the core framework to programmatically call into modules and retrieve their module actions. Other new features based on optional interfaces included content indexing, import, and export. In each of these cases, the core framework could rely on modules to provide content in a specific format that then allowed the core framework to provide advanced portal services.

After multiple beta releases (some of which were deemed not fit for public consumption), DotNetNuke 3.0 was officially released on March 12, 2005. Although there were breaking changes between DotNetNuke 2.0 and DotNetNuke 3.0, a number of modules were immediately available for DotNetNuke 3.0 due to the success of a pilot program named “30 for 3.0.” This program was the shrewd strategy of Scott Willhite and allowed a serious group of commercial module developers to have early access to beta releases of the DotNetNuke 3.0 product, enabling them to deal with any compatibility

issues before the core framework became publicly available. Aside from the obvious benefits of having “applications” immediately available for the new platform, this program also provided some excellent business intelligence. It proved one of Scott's earlier assumptions that the vocal forums community represented only a small portion of the overall DotNetNuke user community. It also exposed the fact that DotNetNuke had found its way into Fortune 500 companies, military applications, government websites, international software vendors, and a variety of other high-profile installations.

DotNetNuke 3.0 was released with two supported languages: English and German. Delivering two complete language packs adhered to one of our newer philosophies of always attempting to provide multiple functional examples to prove the effectiveness of a particular extensibility model. Before long, community members began submitting new language packs in their native dialects that were posted on the dotnetnuke.com site for download. The total number of supported language packs soon surpassed 30. This resulted in incredible growth and adoption for the DotNetNuke framework on an international basis.

Release Schedule

A common open source concept is referred to as “release early, release often.” The justification is that the sooner you release, the sooner the open source community can validate the functionality, and the sooner you get feedback—good and bad—which helps improve the overall product. This concept is often combined with a “public daily build” paradigm, where continuous integration is used to automatically build, package, and publish a new application version every day. These concepts make a lot of sense for single-purpose applications, that is, applications that have closed APIs and have no external dependencies. But plug-in platforms such as DotNetNuke possess a different set of requirements, many of which are not complementary with the “release early, release often” model.

Consider the case of any entity that has developed plug-in resources for the DotNetNuke framework. These could include modules, language packs, skins, or providers. Every time a new core version is released, each of these resources needs to be validated to ensure that it functions correctly. In many cases, this involves extensive testing, packaging a new version of the specific resource, publishing compatibility information, updating related documentation, communicating availability and/or issues to users, servicing compatibility support requests, updating commercial product listings, and so on. You must also consider the issues for the resource consumer. Consumers need to feel confident in the acquisition and installation of application resources. They are not keen on analyzing complicated compatibility matrices to manage their investment. And resellers such as hosters represent an even larger superset of application consumers. The effort involved to perform application upgrades becomes more complicated and costly as the release frequency increases. This is clearly a case where “release early, release often” can lead to issues for framework consumers and suppliers.

For these reasons, DotNetNuke has always tried to follow a fairly well-structured release cycle. This has resulted in fewer major public releases but a much higher-quality, more stable, core application. In general, it has enabled DotNetNuke resource suppliers and consumers to participate in a functional product ecosystem. However, as the number of serious platform adopters increased, so did the demands for better core-release communication.

DotNetNuke Projects

One of the goals of the DotNetNuke 3.0 product release that had tremendous value for the community at large was the abstraction of the modules that were traditionally bundled with the core framework. The core modules were neglected in favor of adding more functionality to the core framework services. This resulted in a set of modules that demonstrated limited functionality and were not evolving at the same pace as the rest of the project. The abstraction of the modules from the core framework led to the formation of the DotNetNuke Projects program: a new organizational concept modeled after the Apache Foundation that allowed many complementary open source projects to thrive within the DotNetNuke ecosystem. From a technical perspective, the modules were abstracted in a manner that conformed to our extensibility model for building “private assembly” modules and allowed each module to be managed as its own independent project. The benefit was that each module could form its own team of developers, with its own roadmap for enhancements, and its own release schedule. As a governing entity, DotNetNuke would provide infrastructure services such as a source code repository, issue tracker, project home page, and email services for the project as well as a highly visible and respected distribution and marketing channel.

Obviously, there are trade-offs that need to be accepted when decomposing a monolithic system into its constituent components, but the overall benefits of this approach reaped substantial rewards for the project. For one thing, it provided a new opportunity for developer participation—basically providing a sandbox where developers could demonstrate their skills and passion for the DotNetNuke project. This helped promote the “meritocracy” model and aided in our Core Team recruitment efforts. The community benefited through the availability of powerful, free, open source components that were licensed under the standard DotNetNuke MIT license. It also allowed the modules to evolve much more rapidly and with more focus than they ever received as part of the monolithic DotNetNuke application. Abstracting the core set of modules was a good start; however, the platform was lacking some other essential modules—modules that were well integrated and provided the common functionality required by most consumer websites. These items included a discussion forum, blog, and photo gallery.

Early in the DotNetNuke 3.0 life cycle, there were discussions with a high-profile third-party software development company that was actively

developing an integrated suite of components with forum, blog, and gallery functionality. Although early indications seemed to be positive regarding collaboration, they unfortunately did not value the opportunity of working with the DotNetNuke community and ultimately decided to instead focus their efforts on constructing their own proprietary solution. Because this decision was not communicated to us until late in the DotNetNuke 3.0 development cycle, it meant that we had to scramble to find a suitable alternative. Luckily, two of our own Core Team members—Tam Tram Minh of TTT Corporation and Bryan Andrews of AppTheory—had been collaborating on a comparable set of modules and had already been offering them for free download to the DotNetNuke community. Discussions with them led to the creation of three powerful new DotNetNuke Projects: the DotNetNuke Forums, Blog, and Gallery.

Integrating third-party modules is not without its share of challenges. An “incubation” period is required to make the module conform to the official DotNetNuke project standards. An official marketing name had to be defined for the project and all references to the old module name need to be updated. This included namespaces, folder names, filenames, code comments, database object names, release package metadata, and documentation. To allow legacy users of the contributed module to be able to migrate to the new DotNetNuke project, a robust upgrade mechanism had to be created. The module also needed to be reviewed to ensure that it does not contain any security flaws or serious defects that could affect the general community. From an infrastructure perspective, the code needed to be uploaded to a dedicated source code repository, an issue tracker project had to be created, and a project home page completed with discussion forum and blog created on dotnetnuke.com. These tasks represented the technical integration issues that needed to be addressed; but an item of even greater importance for third-party modules was management of the associated intellectual property.

Intellectual Property

There are two main contributing factors when it comes to intellectual property: copyright and licensing. The copyright holder is the person who owns the rights to the intellectual property. Normally this is the creator; however, copyright can also be transferred to other individuals or companies. The copyright holder has the right to decide how his intellectual property can be used by others. When it comes to software, these usage details are generally published as a license agreement. License agreements can vary a great deal depending on the environment, but they generally resemble a standard legal contract, explicitly outlining the rights and responsibilities of each party. Copyright holders also have the right to change the license for the intellectual property at their discretion. It is this scenario that requires the most due diligence when dealing with third-party contributions.

Anybody who contributes source code to the DotNetNuke project must submit a signed Contributor License Agreement. This document ensures that the individual has the right to contribute intellectual property to the project without any type of encumbrance. It also transfers copyright for any contributed intellectual property to the project. This is important because DotNetNuke needs to be able to ensure all of its intellectual property is licensed consistently throughout the entire application. It protects the community from a situation where an individual copyright holder could change the license restrictions for a specific piece of intellectual property, forcing the entire community into a reactive situation (a situation we have already seen multiple times in the still nascent Microsoft open source community).

In the case of third-party modules that are fully functional applications with an existing and active user base, the intellectual property rights are owned by the external party. Under this scenario, we could adopt the intellectual property into the DotNetNuke project because it would mean that we would have no control over its licensing. Even if the contributor agreed to license the intellectual property under a complementary MIT open source license, the original copyright holder would still have the ability to change the license at any time in the future, which would put all users of the module in jeopardy. To mitigate this risk, we required that DotNetNuke must have sufficient rights to the intellectual property so that the community is adequately protected. However, we did not feel it would be fair to force

contributors to release all of the rights to their own intellectual property. Therefore, we created a Software Grant Agreement that provides both parties with full copyright to the specified intellectual property. Essentially this means that the intellectual property was split into two independent versions. The contributor owns one version and is allowed to license it or modify it as he or she sees fit. DotNetNuke owned the other version and licenses it under the standard DotNetNuke MIT License for distribution and enhancement. The end result is a win-win situation for both parties as well as the community.

Marketing

The success of any serious initiative must begin with the formulation of specific goals and the ability to measure progress as you work toward those goals. In terms of measuring the growth of the DotNetNuke project, we had traditionally monitored the total number of registered users on the dotnetnuke.com website, the number of new users per month, and the number of downloads per month. These metrics revealed some definite trends but were rather myopic in terms of providing a relative comparison to other open source or commercial products. As a result, we looked for some other indicators that we could use to measure our overall market impact.

SourceForge was the world's largest development and download repository of open source code and applications. Early in its project history, DotNetNuke had established a presence on SourceForge.Net (<http://sourceforge.net/projects/dnn> as shown in [Figure 1.9](#)) and continued to leverage its mirrored download infrastructure and bandwidth for hosting all project release packages. Because SourceForge.Net contained listings for all of the largest and most successful open source projects in existence, it also provided a variety of comparison and ranking statistics that could be used to judge activity and popularity. This seemed to be another good KPI to measure the project's impact in the open source realm. In April 2005, the DotNetNuke project had an overall project ranking of 1,271.

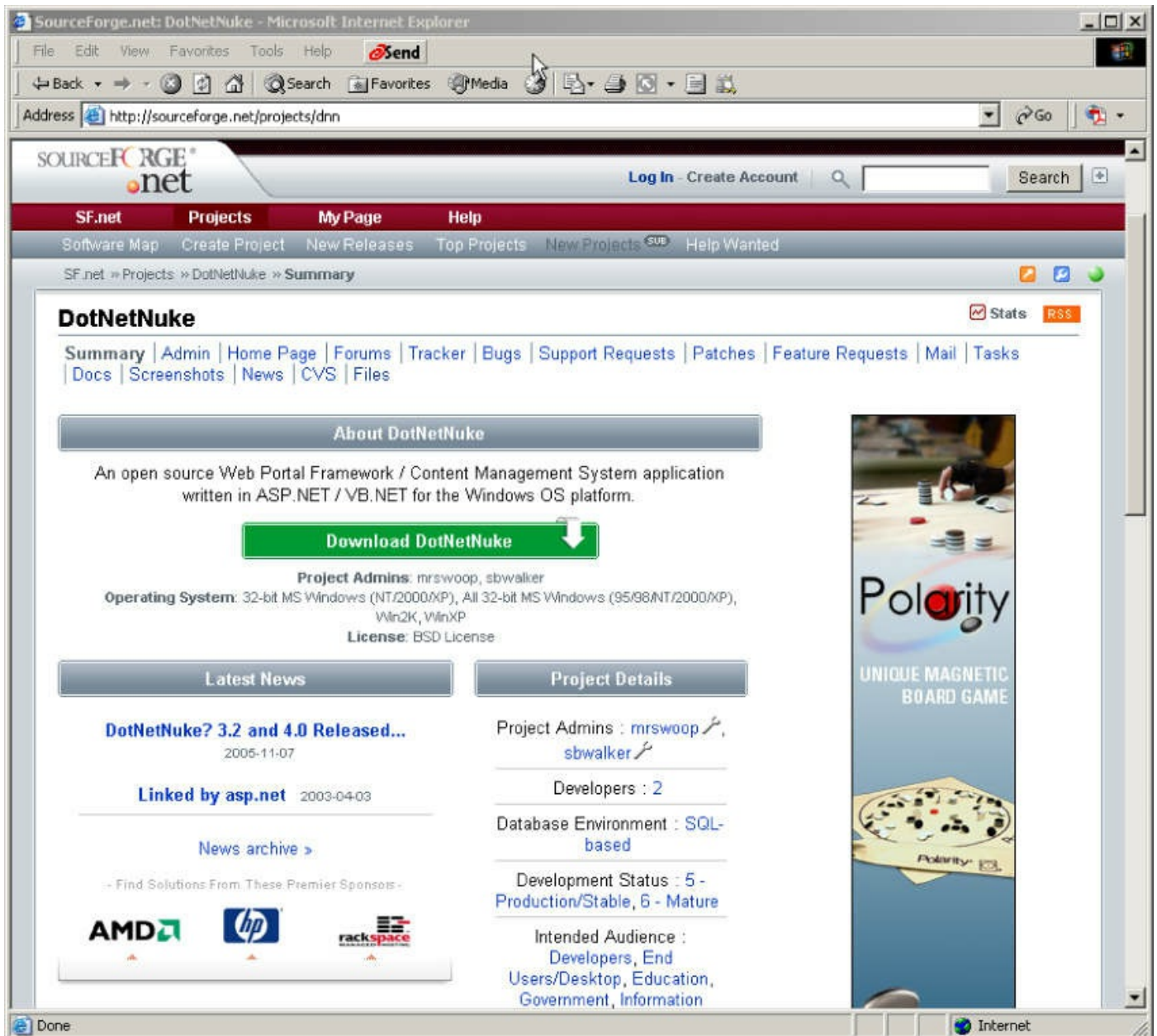


Figure 1.9

One of the items that had been neglected over the life of the project was the dotnetnuke.com website. It had long been a goal to build this asset into a content-rich communication hub for the DotNetNuke community. Patrick Santry made some early progress in this area but recently found his volunteer time diminishing due to personal and family commitments. Because a website is largely an extension of product marketing (another function that had long been ignored) the dotnetnuke.com website suffered from sparse content, poor organization, and inconsistent focus. After the release of DotNetNuke 3.0, a significant effort was invested in improving all aspects of the website. Much of the initial improvements came as a result of evaluating

websites of other open source projects. After extensive deliberation, we decided to organize the site information into three functional areas: user-oriented information, community collaboration, and developer information. New “sticky” content areas were added for project news and community events. The Home Page was completely revamped to provide summary marketing information and project metrics.

In March 2005, another significant milestone occurred in DotNetNuke history. Dan Egan, a passionate DotNetNuke community member, wrote a book for Packt Publishing entitled *Building Websites with VB.NET and DotNetNuke 3.0*. This was the first book published about DotNetNuke and was essential in proving the demand for the product, paving the way for future DotNetNuke books from a variety of other publishers. In addition, a handful of Core Team members, including me, were also collaborating on a book for WROX Press during this time frame, but the demands of getting the DotNetNuke 3.0 product ready for release forced us to slip the publication date. Regardless, any technical content that makes it to mass publication through traditional channels lends an incredible amount of credibility and equity to the project or technology for which it is written. In addition, books can have a positive marketing impact, especially if they reach wide circulation through online retailers and brick-and-mortar bookstores.

In May 2005, Core Team member Jim Duffy was successful in securing a DotNetNuke session on DotNetRocks!, an Internet radio talk show hosted by Carl Franklin and Richard Campbell. This was our second appearance on the show (the first being in August 2004), and it was a lot of fun to talk about DotNetNuke in such a relaxed and open atmosphere. The show focused on the recent DotNetNuke 3.0 release and proved to be a great way to promote some of the incredible new application features. It is hard to estimate the impact of the appearance on the DotNetRocks! show, but it certainly made me a firm believer in the benefits of podcasting as a powerful broad distribution marketing medium.

Microsoft Hosting Program

Throughout the month of May 2005, Microsoft launched the aforementioned hosting program. The purpose of the program was to encourage shared hosting providers to take advantage of Windows technology to grow their hosting businesses. The primary benefit of this program was the Service Provider License Agreement (SPLA), which allowed hosting companies to avoid large capital expenditures and pay their licensing fees based on actual usage. This lowered the barrier of entry in terms of cost and provided a risk-free model to test the demand for services. In addition to the SPLA, Microsoft recognized the value of end-user applications and included substantial promotion of DotNetNuke in the hosting seminars encompassing 30 cities around the world. I was fortunate enough to attend the first seminar in Redmond, Washington, which provided an excellent opportunity to network with the Microsoft Hosting Evangelists, a group of hard-working individuals who were dedicated to the growth of Windows web hosting on an international basis. At the beginning of June, I was also privileged to attend a WSHA seminar in Amsterdam, Netherlands. The invitation was extended by Microsoft Europe, which was especially interested in the localization capabilities of the DotNetNuke application. This trip gave me a deeper understanding of the localization challenges of the international community and also provided me the opportunity to meet Geert Veenstra and Leigh Pointer—two Core Team members who actively participated in and evangelized DotNetNuke since its creation.

Although the Microsoft Hosting program did not reap any direct financial rewards for DotNetNuke, it provided a number of powerful benefits. It exposed the application to an influential group of organizations: large-scale web hosting companies that dominate the shared hosting market in terms of customer base and annual revenues. Companies such as GoDaddy, Pipex, and 1and1 began offering DotNetNuke as part of their Windows hosting plans. The hosting program also caught the attention of the largest hosting control panel vendors. Companies such as SW-Soft (Plesk), WebHostAutomation (Helm), and Ensim added integrated installation support for the DotNetNuke application within their control panel applications. All of these strategic partnerships exposed DotNetNuke to a much larger consumer audience and would not have been possible had it not been for the Microsoft Hosting program.

Collaboration with web hosts also resulted in new application features that were added to satisfy some of their specific business requirements. The ability for DotNetNuke to run in a web farm environment was one such feature that really addressed the application scalability questions beyond a single web server configuration. Dan Caron stepped up yet again to champion these enhancements, producing an architecture with two different caching providers to satisfy the widest array of use cases. Charles Nurse also completed the abstraction of all modules into isolated components that could be optionally installed and uninstalled from the core framework. This change provided additional flexibility for web hosters in terms of being able to customize their offering for clients.

Infrastructure

One of the benefits of the original sponsorship agreement with Microsoft was a free shared hosting account on the servers managed by the ASP.NET team at OrcsWeb. This arrangement served us well in the early stages but the fact that we had extremely limited access (that is, FTP) to the account and absolutely no control over the associated infrastructure services eventually created some challenges for the project. In addition, we had long been leveraging services from PortalWebHosting for back-office items such as DNS, source control, issue tracking, and email, but a recent change in ownership created some friction in regard to legacy promises and agreements. Approaching premium hosting provider MaximumASP in the fall of 2004, we were able to secure a generous formal sponsorship agreement that paved the way for a more centralized and professionally managed project infrastructure.

Initially, MaximumASP provided us with two dedicated servers and a Virtual Private Server (VPS) account on a shared server. One of the dedicated servers was configured as an SQL Server database server and the other as a back-office server. The VPS account was provisioned as a web account for our public website. This configuration served us well initially, but the rapid growth of membership and the lack of control over the web server soon forced us to look for other options. Further discussions with MaximumASP resulted in the allocation of a dedicated web server for our public website. The combination of a dedicated web server and a dedicated database server proved flexible enough to handle our full website requirements. It was not until we added discussion forums to our site and pushed our traffic past 4 million page views a month that we felt the need to consider a web farm configuration.

The physical abstraction of the core application into a more modular organization had a direct impact on our back office project infrastructure. Rather than simply managing a single source code repository and issue tracking database, we now had to deal with many Project sandboxes—each with its own membership and security considerations. In addition, establishing effective communication channels for different stakeholder groups was critical for managing the project. This is one of the reasons why the DotNetNuke Forums Project played such a significant role in the evolution of the DotNetNuke projects. It allowed for a variety of discussion forums to be created, some public and some private, providing focused communication channels for project members.

During 2005, Scott Willhite also made some huge contributions to the project in terms of infrastructure management. In a project of this size with so many active participants, there is an incredible amount of administrative work that goes on behind the scenes to keep the project moving forward. As most people know, administrative tasks are largely unappreciated and only seem to get attention when there is a problem. Scott did his best to keep the endless stream of infrastructure tasks flowing, receiving little or no recognition for his efforts but playing an instrumental role in the success of the DotNetNuke project.

Branding

One of the things that became obvious during the writing of *Professional DotNetNuke ASP.NET Portals* (Wiley Publishing, Inc., 2005) was that our branding message was not clear. Although our trademark and domain name reflected “DotNetNuke,” our logo contained an abbreviated terminology of “.netnuke.” This led to confusion for authors of the book as well as the publisher in terms of what was the correct product branding. As I mentioned earlier in this chapter, the initial branding was constructed with little or no foresight; therefore, it came as no surprise that a major overhaul was necessary.

Initial conversations within the Core Team offered some interesting and sometimes surprising opinions on the DotNetNuke brand. When discussion came to a stalemate, the topic was raised in the public forums that resulted in a similar scenario. Some folks considered the “nuke” term to be too offensive, unprofessional, or shocking to be used as a serious brand name. Others placed a significant metaphorical value in the current logo, which contained a gear embossed with a nuclear symbol. Some preferred a transition to the “DNN” acronym that was often used as a shorthand reference in various communication channels. Further debate ensued over the category we occupied (portal, content management system, framework, and so on) and the clear marketing message we wanted to convey.

As the project founder, I had my own opinions on the brand positioning and ultimately decided to resort to an authoritarian model rather than a committee model so that we could make a decision and move forward. From my perspective, when it comes to technology companies, there is a lot of acceptance for nontraditional brand names (consider Google, Yahoo!, Go Daddy, and so on). In addition, due to the press coverage of the Microsoft Hosting program, the DotNetNuke name achieved a significant amount of exposure; therefore, a complete change in brand would impose a serious setback in terms of brand acceptance and market reach. Taking into consideration the valued perspectives of the Core Team and community, I felt there should be a way to provide a win-win solution for everyone.

I first tried working with a local design company (the same company that produced the DotNetNuke 2.0 site skin), and although it had a real talent for brand identity services, there were no concepts produced that really grabbed my attention or satisfied my goals for the project. Perhaps I was being overly

critical in my judgment of various designs, but I knew that I absolutely did not want to settle for a concept unless I thought it met 100 percent of my criteria. Although Nik Kalyani had been on the Core Team for eight months and had even expressed a serious interest in the marketing activities of the project, it was not until the rebranding exercise where his talents were truly exemplified.

Nik and I started an offline dialogue where we quickly established some complementary goals, at least at a conceptual level. The basic decision was that we wanted to retain the full “DotNetNuke” brand name and strengthen rather than dilute its brand emphasis. We also wanted to reduce or eliminate the negative imagery associated with the nuclear warning symbol in the current logo. Although the abbreviated form of the word “nuke” tended to evoke a negative response from the general population (relating it to bombs and radiation), the expanded form of “nuclear” and “nucleus” had a much more positive response (related to science, energy, and power). The word “nucleus” also had some complementary terms associated with it such as “core,” “kernel,” and so on that worked well with the open source project philosophy. The trick was to find a way to emphasize one aspect over another.

Nik spent countless hours designing alternative logo concepts. From a typeface perspective, he suggested using the Neuropol font, and I really liked the fact that it had a strong technical overtone but not so much that it could not be used effectively in other mainstream media applications. To achieve a uniform appearance for the typeface, we decided to use all capital letters even though the standard format for the brand name in regular print would continue to be mixed case. Nik included a unique customization for the “E” and the “T” letters that resulted in a distinctive, yet professional, styling for the word-mark contained within the logo.

Creating the graphical element for the logo was a much bigger challenge because we were looking for a radically new design that exemplified so many diverse project attributes. To summarize some of the more important criteria, we were looking for something simple yet distinctive, with at least some elements that provided a visual reference to the old logo for continuity. It needed to be scalable and adaptable to a wide range of media (both digital and print) and cost-effective to reproduce. And perhaps the most subjective item I promoted was that the logo should be stylish with my acceptance criteria being, “Would my wife permit me to wear clothing embossed with the logo when we went out in public together?” Nik created more than 40 unique logo

concepts before arriving at a design that seemed to catch the full essence of what we were trying to accomplish (see [Figure 1.10](#)). After working at this for so long and dealing with the discouragement and frustration, it was a euphoric moment to discover the proverbial “love at first sight.”



[Figure 1.10](#)

It is amazing how many diverse concepts can be represented in a single image. The saying “a picture is worth a thousand words” is cliché, but in this case, it certainly summarized the final product. The new logo had the basic shape of a nuclear atom. The nucleus of the atom was shaped like a gear to retain its heritage to the previous project logo. The logo was two basic colors—red and black (using shades of gray to achieve a 3-D effect)—making it much more adaptable and simple to reproduce in a wide variety of media formats than the previous logo (which used shadows and gradients for 3-D effects). The gear had 12 teeth (a number considered to be lucky in many cultures). The intersection of the three revolving electron trails (referred to as the “triad”) could still be subtly viewed as a nuclear symbol reference. With some creative inference, they could also be viewed as the three-letter project acronym: DNN. Later, someone on the Core Team mentioned that the triad bore some resemblance to the Perpetual Motion Interactive Systems Inc. “infinity” logo, a reference I had never formally recognized but something that I am sure played a subliminal role in my selection.

In terms of brand acceptance, we realized there may be significant community backlash related to the new creative brand, especially from companies who were currently leveraging the existing DotNetNuke branding in their marketing materials. Therefore, we were pleasantly surprised at the overwhelming positive feedback we received regarding the new brand identity. Our goal was to roll out the brand in progressive stages with the DotNetNuke 3.1 product release representing the official brand launch to the general community.

With the creative elements out of the way, it was time to finalize the rest of the branding process. Because DotNetNuke serves many stakeholder groups,

it was difficult to come up with a product category that was focused but not too limiting in scope. From a marketing perspective, the board agonized over the optimal brand message. “Content management” was a powerful industry buzzword, but if you compared the capabilities of DotNetNuke in this area with other enterprise software offerings, it became obvious that it would be some time before we could be considered a market leader. The term “portal” had been so overused in recent years that it became severely diluted and lost its clarity as an effective marketing message. Conversely, the emerging term “framework” began to surface more regularly and was starting to gain industry acceptance with both developers and management groups as a powerful software development category. Because DotNetNuke's architectural principles were predicated on simplicity and extensibility, the framework category seemed to be a natural fit. The next step involved clarifying the type of framework. DotNetNuke was primarily designed for use in a web environment and its breadth of features made it well suited for building advanced data-driven Internet applications. The resulting “web application framework” was an emerging industry category in which DotNetNuke could take an immediate leadership role. Where applicable, we could also leverage our “open source” classification to emphasize our community philosophy and values.

One of the toughest parts of any rebranding exercise involves updating all existing brand references to reflect the new identity. In DotNetNuke's case, this affected the content and design of the dotnetnuke.com website, the marketing references in the DotNetNuke release package, and all technical and user documentation. Compared to the time it took to construct the new logo, the time it took Nik Kalyani to create a new site design was minimal (which is truly amazing considering the amount of time and effort that typically goes into a custom site design). I had long been a fan of Nik's minimalist style, which emphasized clean presentation, lightweight graphics, and plenty of whitespace. Nik's expert grasp of the DotNetNuke skinning architecture enabled him to create a combination of skins and containers that were applied in a matter of minutes to completely transform the entire website. The new site design was creative yet professional and eliminated the “cartoonish” criticisms of the previous site design (see [Figure 1.11](#)). Nik also created our first professional document templates that would provide consistency and emphasis of our branding elements within our technical and user documentation.



Figure 1.11

Tech Ed

At the beginning of June, there was a massive Microsoft technology conference, Tech Ed, in Orlando, Florida. Based on a generous invitation from the International .NET Association (INETA), Scott Willhite and I were provided with an opportunity to attend the event as special guests. The timing was perfect because *Professional DotNetNuke ASP.NET Portals* was officially released at this event, as was the new project branding. Joe Brinkman and Dan Caron were able to attend some aspects of the book launch festivities, and we managed to jam a substantial amount of marketing activities into the five-day event. We had a dedicated Birds of Feather session, two community focus sessions at the INETA booth, a guest appearance at an INETA User Group workshop related to building effective websites (where we learned 90 percent of .NET user groups were already using DotNetNuke), and a number of book signings scheduled by WROX Press at the Tech Ed bookstore. The DotNetNuke book was the top-selling developer book at the Tech Ed bookstore for the event—a fact that emphasized the growing popularity of the project. We also distributed official DotNetNuke T-shirts that showcased the new project branding, a popular item amid all the typical free swag provided at these events.

Seizing the opportunity of having the majority of the DotNetNuke board of directors together in one place, we had our second official board meeting, an all-day session in the conference room of our hotel in Orlando. On the agenda was a serious discussion related to Core Team reorganization and key project roles. For quite some time, we had realized that the current flat organizational structure was somewhat dysfunctional and that we ultimately needed more dedicated management resources to accomplish our goals. However, to support these resources, we needed a sufficient financial model. Discussion focused on the pros and cons of various revenue opportunities, their revenue potential, and their perceived effect on the community ecosystem. We also talked about what it would take for the current board members to commit to full-time dedicated roles in the organization and the associated financial and security implications. A lot of really deep discussion ensued, which gave us a much better mental picture of the challenges that lay ahead if we truly wanted to take the project to the next level.

Following the publication of *Professional DotNetNuke ASP.NET Portals*, there was a bit of a media frenzy around the relationship between Microsoft

and the open source phenomenon. Some of my personal opinions and quotes from the book found their way into an article published on CNET (one of the leading mainstream news sites), resulting in a lot of additional exposure for the project. It was interesting to see the power of the media at work, where a reference in a highly visible and trusted journalism channel can lead to broad distribution of a particular message (much like a stone in a pond leads to a concentric series of expanding ripples). For the most part, large companies are the most successful at leveraging these medial channels, but special-interest organizations also have the opportunity to make a significant impression.

Credibility

Although DotNetNuke had experienced a healthy growth rate through its open source philosophy, it had largely done so by appealing to the needs of grass-roots developers. Although these stakeholders represent an integral part of the high-tech marketplace, there is another group that is far more influential in terms of market impact. The so-called “decision-makers” represent the management interests in serious enterprise-level business organizations. For DotNetNuke to make the transition from a developer-oriented open source project to a serious enterprise software contender, it needed to appeal to the decision-maker.

Where developers think in terms of short-term technical decisions (that is, “What tool can I use to get this job done as quickly as possible so that I can impress my boss?”), decision-makers think in terms of long-term business decisions. They are interested in the future support of a platform or product. They consider solutions in terms of “investments,” “security,” and how much “risk” is associated with adopting a particular technology as part of their company infrastructure. And regardless of the technical superiority of a software solution, the adoption criteria always come down to basic trust and consumer confidence. So the challenge for an open source project like DotNetNuke is establishing the necessary level of credibility to be taken seriously.

In the commercial world, customers get a sense of confidence based on the fact that they have paid licensing fees to a vendor that generally provides them with a certain level of future support. Obviously nothing is guaranteed, but this financial model provides both parties with a sense of security and responsibility. Another thing that the financial model affords is the ability to market the product through traditional channels—channels that decision-makers tend to monitor on a regular basis.

In the open source world, there are no licensing fees, which helps contribute to the lower cost of ownership but also leaves the investment/security aspect somewhat lacking. If you look at Linux, for example, you will notice that the broad industry buy-in for the operating system did not occur until after some serious market vendors (Sun and IBM) pledged their support. As soon as this happened, many medium-large companies began to take Linux more seriously. And this was not because Linux received any product improvements through these relationships, but rather because it reduced its

risk perception in the general marketplace. And without traditional licensing fees, open source products generally do not have the budget to leverage traditional marketing channels and must instead rely on grassroots and viral marketing techniques.

So let's consider some of the ways in which an open source product can improve its credibility and reduce its risk perception for decision-makers. Clearly one way is that it can align itself with large, respected vendors who lend credibility (that is, "If vendor X thinks it's good, then so do we"). Another way is to have mainstream books, magazines, and mass media distributors publish information about the product, contributing to the overall community knowledge base and providing recognition. Yet another option is to identify reference implementations that exemplify the best qualities of the product and impress people with their performance, elegance, or extensibility.

Another way is to demonstrate a proven track record and history for supporting the community, especially through platform transitions where the likelihood of project failure is high. The overall size of the community ecosystem, including the open source participants, consumers, and third-party service providers, is another critical aspect in demonstrating credibility.

DotNetNuke definitely made some significant advancements in credibility in 2005. The strong working relationship with Microsoft reaped rewards with the hosting program. The publication of *Professional DotNetNuke ASP.NET Portals* by Wiley Publishing, Inc. and *Building Websites with VB.NET and DotNetNuke 3.0* by Packt Press provided some excellent recognition through traditional publishing channels. Articles and references in mainstream magazines such as *Visual Studio Magazine*, *ASP.NET Pro*, *CoDe Magazine*, and *.NET Developers Journal* also provided some great benefits. The showcase on dotnetnuke.com contained many diverse reference implementations, and we had proven through three years of product upgrades that we were committed to supporting the community. The membership and download metrics continued to grow exponentially, as did the number of independent software vendors (ISVs) providing products or services within the DotNetNuke ecosystem.

Trademark Policy

Unfortunately, an unexpected issue arose in the summer of 2005 that immediately put the project into crisis mode. Based on some invalid assumptions, a software consultant from Australia recommended that his client register a trademark for the DotNetNuke name in Australia. Aside from the obvious ethical implications, the immediate reaction was that this move was based on ulterior motives that could potentially hold the entire Australian DotNetNuke community hostage. Further communication revealed that the Australian company had concerns over the official trademark registered in Canada, specifically in regard to the fact it was embedded within the application source code and binaries and that its business investment could be compromised if restrictions were ever put on trademark usage. Ultimately this whole situation revealed a number of critical issues when it comes to trademarks. First, the holder of the trademark must publish a policy that clearly defines the allowable usage of the mark under a wide range of use cases. Second, the trademark holder must make every attempt to enforce the policy so that the mark does not become a common term and lose its value as a protected asset. Third, a trademark must be registered in every jurisdiction where it intends to be used.

To satisfy the first requirement, I firmly believe in the philosophy of “standing on the shoulders of giants.” Research revealed that Mozilla had recently gone through a similar project challenge, so we decided to use its recently published trademark policy as a template for our own. The political ramifications of introducing the policy at this point seemed controversial but absolutely necessary if we intended to protect our brand. After extensive research, review, and legal advice, we finally announced the trademark policy in conjunction with the logo guidelines in July 2005. The overall community feedback was quite positive because the policy made every effort to emphasize our open source roots and strong community ideals.

To satisfy the second requirement, all marketing materials were updated to reflect the trademark policy guidelines, and many community sites made changes to bring their use of the trademark into compliance. We also obtained legal advice on the creation of a Trademark License Agreement to be used in situations where third parties required the right to use the DotNetNuke trademarks for specific business purposes.

The third requirement was somewhat more challenging to deal with because

it had substantial financial implications. The cost to register an individual trademark in a specific jurisdiction (country) can cost anywhere from \$2,000 to \$5,000. As an organization, we simply do not have the financial means to support such a large expenditure. So instead of considering all jurisdictions, we decided to focus on those jurisdictions that had a large project following. These included the United States, Canada, Australia, Japan, and the European Union. This whole experience gave me a much deeper understanding of the financial commitment required by large multinational companies that want to protect their brand around the world.

ASP.NET 2.0

In July 2005, we recognized that we had approximately four months to prepare for the launch of Microsoft's next-generation software development platform. ASP.NET 2.0 had been under development for three years and had finally reached the point where it was ready for public release. Aside from reading the standard marketing propaganda in the various trade magazines catering to the Windows platform, I had not done significant research into the specific challenges DotNetNuke faced as a product related to this platform upgrade. And, as is usually the case, we quickly found out it was going to be some of the unpublicized platform changes that were going to cause us the most difficulty.

Based on early community feedback for the ASP.NET 1.0 release, Microsoft decided to completely overhaul the way web projects operated, including substantial changes to the underlying compilation model. Because DotNetNuke's advanced modular architecture strayed so far from the traditional monolithic ASP.NET application model, these platform changes had a significant impact on the project. Our solid working relationship with Microsoft reaped benefits in that we were able to engage in some focused dialogue and onsite meetings in Redmond with the Microsoft product managers who understood the nuances of the new ASP.NET 2.0 platform better than anyone. Scott Guthrie, Simon Calvert, Omar Khan, and a number of other key Microsoft resources got personally involved in assisting us to find a suitable migration path.

I have to admit I was a vocal critic during these early discussions because I could not understand the business cases that precipitated some of the major architectural changes. But after working closely with the Microsoft product managers, I began to warm up to the benefits of the new model and started to envision how we could leverage its capabilities to expose some powerful new options to the DotNetNuke community. But before we could focus on these new options, our most critical requirement was that we could not have breaking changes in the DotNetNuke framework in our ASP.NET 2.0 release. The main business criteria driving this requirement was the fact we had just had a major release with significant breaking changes in March 2005, and we could not risk an all-out community revolt (or product fork) based on compatibility issues.

Research and discussion proceeded throughout the months of July and

August as we worked with Microsoft to find an optimal solution. Feedback from the community seemed to be mixed. People who were victims of the Microsoft propaganda machine seemed to think that the release of ASP.NET 2.0 would signal the end of DotNetNuke, because it promised to deliver so many overlapping application features. Other people who had adopted DotNetNuke as part of their business infrastructure expressed apprehension and fear regarding ASP.NET 2.0, based on their past experience that a significant platform upgrade usually resulted in a costly migration effort. Surprisingly, out of all the feedback collected, it appeared that nobody was making a serious attempt to perform the upgrade on their own and that they were waiting for us to provide a migration path (as we had always done in the past). This element of trust was not lost on me, and I did my best to blog on a regular basis to provide public communication of our progress.

Reorganization

Throughout the summer and fall of 2005 there was ongoing discussion related to Core Team reorganization. Based on the guidelines that had been created when individuals were invited to join the team in the summer of 2004, there was clearly a group of members who had not lived up to their commitments. The list of responsibilities included staying involved in Core Team business through the private discussion forum; participating in weekly Core Team chats; contributing bug fixes, enhancements, or documentation to the core product; and being active in community support channels. There were many legitimate reasons, both personal and business-related, which led to inactivity for team members. However, the unfortunate side effect is that it led to a community perception that based on the total number of Core Team members, we were underachieving in terms of our capabilities as a whole. The Core Team reorganization meant that a number of team members needed to be retired to make way for some new members who had earned the right to participate based on their community accomplishments over the past year. The project had never had to deal with a situation like this in the past, and it's safe to say that as software developers, we are much more adept at solving technical problems than human-resources issues. So the dilemma was how to break the news to the inactive members in a professional and courteous manner that still respected their past accomplishments and left the door open for future participation. It was Scott Willhite who demonstrated the most experience and wisdom in this area, as we worked on establishing effective human resources processes for the organization.

Since the original formation of the Core Team, all members had received equal rights in terms of project participation. This included not only communication channels but also permissions to the product source code repository. This model worked well when the team was small and all members were on equal footing in terms of their technical abilities. However, it proved to be a challenge when the team grew in size and members were added with varying technical backgrounds. DotNetNuke had grown into a mission-critical web application framework that many businesses now relied on for rock-solid performance and reliability. We could no longer accept the risk of inexperienced team members checking in code that could compromise the stability of the application. As a result, we needed to refactor our project roles to reflect the new project requirements.

A common theme that helped drive the refactoring of the project roles was accountability. In the past, we had witnessed the fact that without accountability, an individual would not exhibit the same level of commitment, dedication, or passion for the project. As a result, it was important to provide Core Team members with areas of accountability where their contributions would be highly visible and easily recognized by the general public. This public aspect provided them with a much greater benefit in terms of visibility in the community, but it also made them a target for criticism if they were inactive because they were personally responsible for specific areas of the project.

Using the Apache Foundation as a meritocracy reference, we made some significant changes to the organizational model of the project. The old “Inner Team” designation was abolished in favor of a new “Core Team Trustee” role. Scott Willhite came up with this new name based on the desire for industry-accepted terminology and the fact that this innermost project role assumed the highest level of trust from a development perspective. Core Team Trustees had multiple years of experience on the project, had successfully demonstrated their technical aptitude, and as a result were granted write access to the core repository. The old “Outer Team” designation was simplified to “Core Team Member”—a role that was able to participate in all Core Team communication channels but was only provided read access to the source code repository. In addition, we added a role for the DotNetNuke Projects of “Project Team Lead.” This role was responsible for managing the project infrastructure and communicating project status to the Core Team.

Microsoft Conferences

The month of September 2005 began with the Professional Developer Conference (PDC) in Los Angeles, California. Based on a kind gesture from Microsoft, a large number of Core Team members were provided with free registration for the event in exchange for analysis of key ASP.NET 2.0 features that could be used in the DotNetNuke framework. Scott Willhite, Dan Caron, Nik Kalyani, Jon Henning, John Mitchell, Charles Nurse, and I were all able to attend the event, bringing together in one place the largest group of Core Team members ever. It was an excellent opportunity to get to know one another, and we spent a lot of time hanging out together, exploring the exhibitor area, hosting a Birds of Feather session, visiting Universal Studios, and attending a variety of conference sessions.

The DotNetNuke board of directors, with the recent inclusion of Nik Kalyani, also took the opportunity to have some serious meetings regarding the progress of the revenue opportunities discussed at TechEd. The summer had not been productive in getting any programs launched other than advertising and sponsorship, and Nik took a lead role in attempting to clarify both our marketing and financial initiatives for the next 12 months. Specific board members were assigned to each major opportunity, and projections were presented and discussed in terms of assumptions, benefits, and execution tasks. We had a lot of work ahead of us, including a major platform transition, now firmly scheduled for November 7, 2005.

Later in September, Microsoft hosted a three-day summit for its Most Valuable Professional (MVP) community members. Based on public achievements, a number of DotNetNuke Core Team members earned this award of distinction in 2005. Bruce Hopkins (Georgia, USA), Phil Beadle (Australia), Cathal Connolly (Ireland), Jim Duffy (USA), and I (Canada) were all able to attend the private summit in Redmond, Washington. The summit provided the opportunity to get to know these Core Team members on a more personal level, including their appetite for social festivities. I was also able to spend some time with a number of prominent ASP.NET personalities and DotNetNuke evangelists whom I greatly respected in terms of their contributions to the community. In addition, there was also a large representation of Microsoft employees at the MVP summit that resulted in some excellent networking opportunities and offline discussions. Steve Balmer's keynote address provided some valuable insight into the roadmap

for Microsoft's products and revealed areas where DotNetNuke could focus its efforts to strengthen its market position in the coming year.

Directly following the MVP summit, I had the privilege of attending my first ASPInsiders summit as well. The ASPInsiders represent a group of well-respected industry leaders in the Microsoft ASP.NET community. I had recently been inducted as an official member and appreciated the opportunity to be included in such an elite group of professionals. Perhaps the most important benefit of being an ASPInsider was that it provided representation for the DotNetNuke development community and validation of our extensive contributions to the industry. Due to its small focused membership, the ASPInsiders summit had a personal and direct interaction with Microsoft employees, allowing its members to provide feedback on a number of exciting new technologies. The networking opportunity was incredible, and the intricate dynamics of the various personalities and companies represented were especially interesting.

DotNetNuke 4.0

Throughout the months of September and October, Charles Nurse was instrumental in working on the migration to the ASP.NET 2.0 platform. He invested a massive amount of time researching compatibility issues, creating various proof of concepts, and communicating regularly with Microsoft. He actually pursued two different agendas simultaneously: the upgrade of DotNetNuke 3.0 to ASP.NET 2.0 from a runtime perspective and the creation of a new web project model for DotNetNuke 4.0 that provided a development strategy for the future.

To support the community, we concluded that we would need to support two parallel code bases for an undetermined period of time: DotNetNuke 3.x (ASP.NET 1.1) and DotNetNuke 4.0 (ASP.NET 2.0). Obviously, a more optimal solution would have been a single code base that worked on both platforms; however, this simply was not possible based on the platform compilation changes in ASP.NET 2.0. In addition, we did not know what to expect in terms of the adoption rate for the new Microsoft platform. Therefore, it seemed natural that we focus on developing for both ASP.NET 1.1 and 2.0 in the short term. An unfortunate side effect of this model involved a general recommendation to develop to the lowest common denominator (that is, not leverage ASP.NET 2.0–specific technology) and synchronizing all fixes and enhancements across the two code bases.

One of the greatest achievements in the platform migration was that we were able to fully satisfy our business requirement for no breaking changes. DotNetNuke modules and skins developed on ASP.NET 1.1 could be installed directly into the ASP.NET 2.0 environment without any changes whatsoever. This had massive benefits for the commercial DotNetNuke ecosystem because vendors could continue developing their modules as a single code base on the ASP.NET 1.1 platform but offer their packaged products for sale in both channels.

The only item that remained outstanding right up until the week before the November 7 launch was how to develop DotNetNuke 4.0 modules on the ASP.NET 2.0 platform. The new dynamic compilation model in ASP.NET 2.0 created some challenges for many of our runtime extensibility features, especially where they relied on object instantiation through reflection. As is often the case with technical problems, the answer is out there—it's just a matter of finding the right person to ask. As luck would have it, a Microsoft

developer (Ting-Hao Yang) who was copied on some of the communication between our team and the Microsoft ASP.NET Product Manager group finally responded with details on a new ASP.NET 2.0 framework method that ultimately solved all of our remaining reflection issues. In the end, all that was required was a change to a single method in the DotNetNuke 4.0 core framework (to use `BuildManager.GetType`).

One of the benefits of the new ASP.NET 2.0 platform was that Microsoft had put a lot of focus on making the technology more accessible to the general developer community. A key deliverable in this strategy was the release of an entire suite of free “Express” tools. Included in the Express line was a tool named “Visual Web Developer” that provided a functional Integrated Development Environment (IDE) for ASP.NET 2.0. Leveraging the benefits of this powerful new tool, we created a DotNetNuke 4.0 Starter Kit that enabled a developer to configure a fully functional development environment within minutes. This had significant implications on the DotNetNuke development community because it lowered the barrier of entry and now made it possible for any aspiring software developer, from beginner to advanced, to be instantly productive with the DotNetNuke web application framework. Combine this with the free SQL Server 2005 Express database engine and you have a zero cost development environment. Visual Web Developer could not be used to develop server controls or class libraries; however, the fact that the DotNetNuke extensibility architecture was based on user controls made it a perfect fit.

Not wanting to neglect the existing DotNetNuke 3.0 community by focusing solely on ASP.NET 2.0 migration, we decided to integrate a few powerful new features that had long been requested by the general community. Core Team member Tam Tran Minh had been developing an Active Directory integration component for a number of years and agreed to contribute it as a fully supported core framework component. Additionally, Jon Henning had been busy working on a full-featured JavaScript API that would allow developers to leverage powerful client-side behavior in their modules. This included a new menu control, the DNN Menu, and an implementation of the popular Asynchronous JavaScript for XML (or Ajax) technology. Ajax technology had become one of the hottest new trends for web development, and it is important to note that DotNetNuke included a powerful Ajax library well before the announcement of ASP.NET Ajax by Microsoft. The combination of these features offered benefits to both platform consumers and application

developers, and further strengthened our core platform offering.

The official Microsoft launch date for ASP.NET 2.0 was set for November 7, 2005. We knew if we could release DotNetNuke 4.0 to coincide with this event, we would be able to ride the huge marketing wave created by Microsoft. Because we had always advocated “releasing software when it is ready,” this hard deadline imposed some serious challenges on our meager project resources. Aside from the obvious technical deliverables, we had communication and marketing deliverables that also needed to roll out in unison. Nik Kalyani showed his ability to pull things together on a tight schedule, and we launched our first monthly newsletter to the entire DotNetNuke registered user base (now 200,000 registered users) on November 7. The response was overwhelmingly positive as the significance of the achievement began to sink in. In the month of November, we recorded 165,000 downloads, far eclipsing any previous monthly download total in the history of the project.

An interesting aspect to consider in the ASP.NET 2.0 migration was that we delivered a fully managed upgrade to users of the DotNetNuke web application framework. Anyone who has ever attempted a major platform upgrade on his or her own should recognize the incredible value of this accomplishment. We had effectively eliminated a budget line item of considerable cost and effort from thousands of IT departments and business entities around the world. Compare this to scenarios where companies create their own custom ASP.NET 1.1 applications. In these cases, each company would need to invest significant resources and funding to work out its own web application migration strategy. Or compare this to another scenario where you adopt another web application framework, commercial or open source, which had not even considered the upgrade challenges posed by ASP.NET 2.0 and were going to force you to postpone your upgrade until it fit their own release schedule. In either case, the decision to adopt DotNetNuke as part of an organization's business infrastructure had certainly paid dividends worthy of the attention of any business decision-maker.

Immediately following the DotNetNuke 4.0 release, we focused on stabilization issues that were exposed through testing by a larger community audience. Another area that received dedicated focus was the Module Item Template feature of the DotNetNuke 4.0 Starter Kit. Through research and persistence, we were able to construct a DotNetNuke Module Template that could automatically create all of the development resources required to build

a fully functional module in DotNetNuke 4.0. It even had some parameterization capabilities so that the template could be customized at runtime to meet the needs of the developer. I wrote an article describing the Starter Kit and Module Template and posted it on the public forums on www.asp.net. The article proved to be popular, with nearly 30,000 views recorded in the six weeks following its publication. It turned out that the changes in ASP.NET 2.0 resulted in some decent productivity benefits for module developers, further improving the capabilities of the DotNetNuke framework.

An interesting event occurred in December 2005, well after the official launch of ASP.NET 2.0. Based largely on the feedback that we provided Microsoft during our product migration efforts, Microsoft announced some add-ons for Visual Studio 2005 that added back ASP.NET 1.1 development support through Web Application Projects as well as compilation and merge support through Web Deployment Projects. Based on its superior architecture and incredible popularity, DotNetNuke was able to unite a significant portion of the Microsoft developer community and create a much stronger voice and more compelling argument in favor of specific platform features than would have ever been possible for individual developers. Besides the fact that these add-ons provided some critical options for web application developers, it was really gratifying to see that our direct feedback could have such an immediate and influential effect on the industry.

Slashdotted

In October 2005, I wrote a blog titled “No Respect for Windows Open Source.” The blog was a political rant based on the fact that because DotNetNuke did not run on a fully open source stack of software components (that is, Linux/Apache/MySQL/PHP or LAMP), it did not get any respect from the general open source community. Further, it argued that all open source projects regardless of platform should be judged solely on the validity of their open source license and ideals. The blog was picked up by Slashdot, the largest independent news site for information technology and resulted in a lot of exposure for the project. The posting on Slashdot generated more than 500 comments, each with a unique perspective on the Windows open source paradigm.

In October, we were approached by *.NET Developers Journal (.NETDJ)* to do a series of articles on the DotNetNuke project. This was an excellent opportunity to showcase various aspects of the project in a mainstream magazine. A number of Core Team members were identified as potential authors and the first article in a series of six was published in the November edition of *.NETDJ*. Forging relationships with publishers is a great way to raise the profile of the project and open doors for future opportunities. In this case, working with SYS-CON (the publisher of *.NETDJ*) reaped rewards in terms of being approved as a featured speaker in the upcoming SYS-CON Enterprise Open Source conference in June 2006.

By the end of 2005, our SourceForge.Net ranking had climbed to #75 (out of all the open source projects in the world). We were consistently getting 15,000 new registered users per month, and our project downloads averaged 120,000 per month. The `dotnetnuke.com` site was now serving 4.5 million page views per month, and every indication was pointing to even more improvement in 2006.

Benefactor Program

As much as there is a romantic notion regarding a distributed group of purely volunteer resources working together in their free time to produce an enterprise-level software product, it does not represent reality. To effectively manage all of the aspects of a professional software product, dedicated management is an absolute requirement. This does not just entail the standard project management principles for software development, but also the legal and marketing aspects of managing a high-profile technology asset. Since the project inception I had been able to commit 100 percent of my time to the project only because there was a sufficient stream of project revenue to support my needs. And throughout the life of the project, a number of team members had been financially compensated for various deliverables so that we could meet obligations and scheduled deadlines. The financial resources came from a variety of sources, including third-party sponsorship, advertising, and custom consulting opportunities. Unfortunately, the revenue streams were not sizable or stable in terms of securing multiple resources for long-term engagements. Essentially, we were trying to operate a product company without any direct product revenue. And with the constant growth of the project, the demands were increasing rather than decreasing, putting even more pressure on the minimal set of project resources.

Back in July 2005, I concluded that without a dedicated sales effort, the dotnetnuke.com website was never going to reach its full potential as a revenue-generating asset. (We had published ad rates on the site months earlier and had not received many serious inquiries.) I decided it was time to more actively cultivate our advertising and sponsorship revenue streams and that it was going to require spending some money to make money. Armed with a huge number of industry contacts collected at Tech Ed, I hired a full-time resource to actively manage the advertising and sponsorship program. Due to major content improvements made in the previous four months, the dotnetnuke.com website became a targeted channel for the Microsoft development community. I hired my brother, Bill Walker, full time to act as the DotNetNuke advertising manager, and despite his lack of knowledge of the product or industry, he hit the ground running. By simplifying the advertising rate sheet and employing traditional sales techniques, we were successfully able to substantially grow this revenue stream in a relatively short time. However, it was still not a model that would scale to supporting the large successful organization we wanted to become.

In the fall of 2005, while driving home from a business trip, I spent some dedicated time immersing myself in the revenue model dilemma. Over the years, I did a lot of research on business models for open source projects, and the big question was, “How do you sustain an open source organization while still adhering to its open source ideals?” There were obviously a number of companies that had demonstrated their ability to succeed in this area by employing a variety of financial options; however, I was keenly aware that each model had its own set of disadvantages.

One of the other recurring themes I kept thinking about is “who we serve.” In a traditional business model, you serve your customers—but this generally assumes that some money is changing hands. For DotNetNuke, I would like to think that our open source community is who we serve, but because they are essentially using the product for free, it becomes challenging when other stakeholders step forward with financial support.

Examining each of the more popular open source revenue models based on this theme proved to be a useful exercise.

A pure volunteer option has no revenue model. As a result, it has no resource cost—but at the same time it has no accountability, responsibility, or dedicated management. It could be argued that although it is supposed to serve the open source community, it really does not because there are no motivating factors driving the development and support.

A dual license model had been effective for a number of open source projects because it allowed for an open source version as well as a commercial version of the same product. This is possible only if the project owner has clear ownership of the copyright for the code. The commercial version provides traditional licensing revenue that helps sustain dedicated management and developer resources, resulting in improved accountability. Unfortunately, it tends to lead to a number of conflict-of-interest scenarios within the ecosystem. For one thing, there is a constant problem of deciding which features belong in the open source version of the product and which in the commercial version. The commercial license often eliminates many of the advantages that are fundamental to a customer choosing an open source solution. Extensibility options are sometimes throttled as the company attempts to control the financial ecosystem around the product. And the company is often forced to show favoritism through support and marketing channels to its paying commercial customers over the organic open source community.

A sponsorship model involves utilizing a revenue stream from one or more third-party funding sources. Although this revenue model results in funding for dedicated management, it often compromises the project ideals as the sponsor attempts to exert its influence over the project roadmap and marketing goals. It also results in a revenue stream that is variable, creating challenges in terms of cash-flow requirements. In addition, the project needs to be extremely diligent regarding the ownership of the intellectual property so as not to put itself in a situation where the third party could sue the project for copyright infringement or affect the open source project licensing.

A professional services model is based on a concept where the platform maintainer does a significant amount of custom consulting for a third-party client. The revenue from the custom consulting is used to fund the dedicated management for the open source product. Unfortunately, this model tends to consume a high level of resources to qualify leads, formulate contracts, manage accounts, obtain signoff, and keep the pipeline full of revenue opportunities. The revenue stream is variable, affecting cash flow, and key project resources are often required to focus on specific client requirements rather than supporting and improving the open source product.

A charitable donations model is a popular concept in the traditional open source world because it involves voluntary community financial support of the project. The problem is that it does not generate a consistent, sustainable revenue stream, which means it is unable to secure dedicated management resources. In addition, there is a tendency for community members to assume that other members are making financial donations, when in reality the project is receiving no financial contributions from anyone.

A vertical application model leverages the open source product to create a highly specialized, commercial, vertical market application. The vertical market application typically generates revenue through an application service provider (ASP) revenue model, which contributes funding back to the open source project. The challenge is that it requires focused management and marketing in the vertical market, complete with domain challenges, competition, legal considerations, and political constraints. The open source application also tends to cater the product roadmap to the needs of the vertical market application, resulting in a less robust application framework.

Because each of the common revenue models has its own set of issues, it made me brainstorm what I would consider to be an optimal open source revenue model. The main criterion is that the project should serve the open

source community (“by the people, for the people”). It should be objective and open, avoiding conflict of interest and adhering to open source ideals. Finally, the revenue stream must be consistent and sustainable, capable of sustaining multiple dedicated resources.

An interesting economics philosophy that Scott Willhite turned me on to was the concept of the “abundance mentality.” In terms of business value, an “abundance mentality” refers to an attitude of growth. Essentially, it means that the overall size of the ecosystem becomes larger as the number of opportunities within the ecosystem increases. By working together with various stakeholders in the ecosystem, all members of the collective group benefit through a greater abundance of revenue-generating opportunities. The opposite of the “abundance mentality” is the “scarcity mentality,” where participants consider the size of the ecosystem to be constrained and the goal is to capture as much of the market share as possible (choking out the smaller competitors in the process). DotNetNuke's extensible architecture and open source philosophy constantly push the envelope in terms of creating new business opportunities within the community. It was another principle that needed to be adhered to in our quest for a suitable revenue model.

With all of these ideas swirling in my head, I concluded that a membership subscription concept could be an effective revenue model for advancing our goals. It would mean that the open source project was funded by the community. It would also mean that the project was accountable and responsible to the community. Through the creation of new benefits, we would be able to provide more opportunities for community members to participate in the project ecosystem. From a public perspective, it would provide a defined method for any supporter, big or small, to contribute to the project. And we would not need to compromise any of our open source ideals. Membership would be available by subscription that would create an ongoing, consistent revenue stream.

The DotNetNuke Benefactor Program (see [Figure 1.12](#)) was officially launched in December 2005. Nik Kalyani came up with the marketing term “benefactor” because it clearly communicated the financial support goal of the program. The program had four levels of participation to cater to the needs of various stakeholders in the community, from individual developers to enterprise business organizations. The initial set of benefits was targeted to each program level, and the administrative aspects of the program were automated as much as possible to provide a seamless user experience. The

overall response to the program was positive and paved the way for future revenue opportunities.

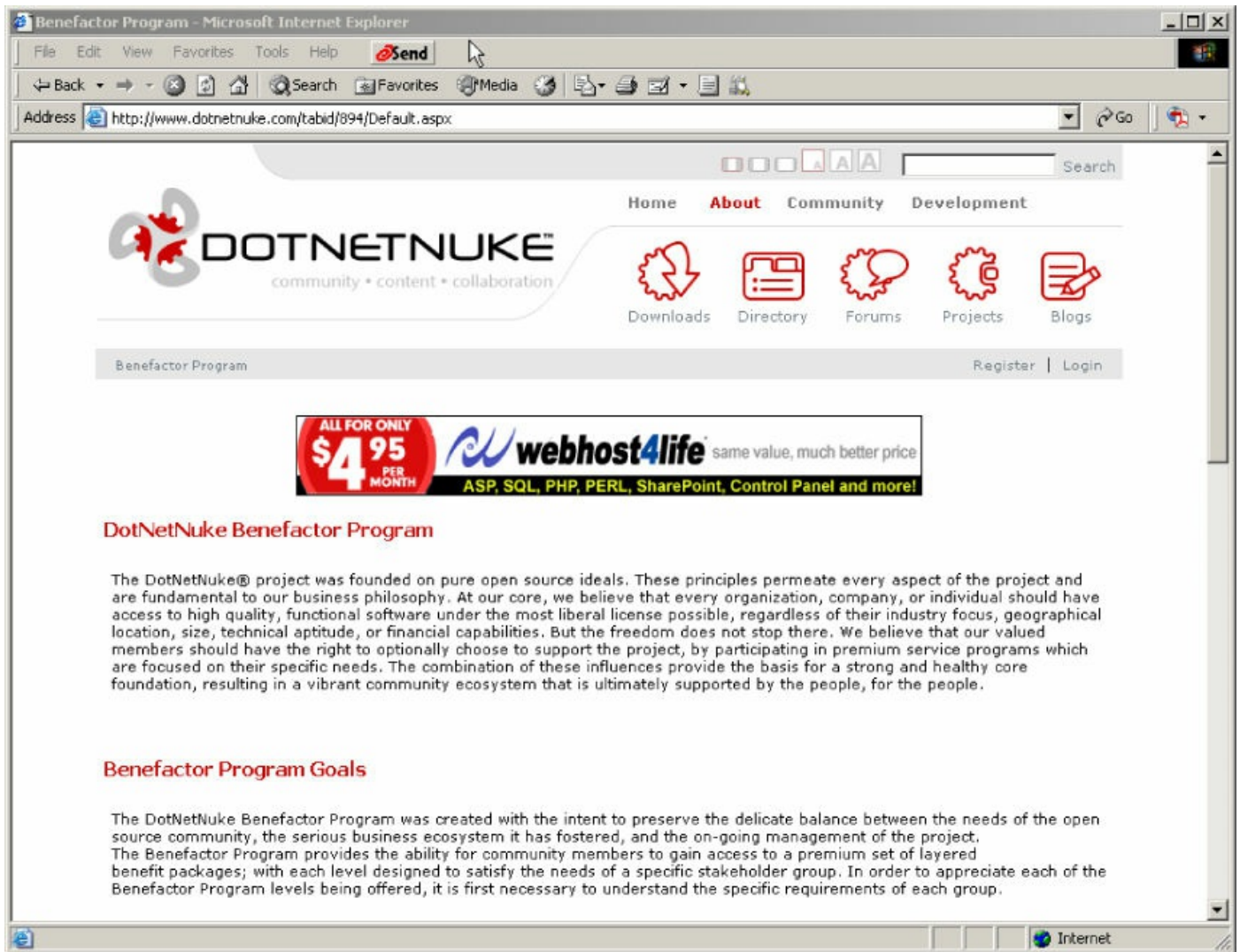


Figure 1.12

Opportunists

In the fall of 2005, DotNetNuke was starting to gain considerable momentum. The open source community was growing, and the commercial ecosystem was becoming stronger and more diverse. DotNetNuke was being used in more enterprise deployments than ever before, and the brand was beginning to become more recognized and respected in the industry. A huge opportunity began to emerge, and this caught the attention of a number of serial entrepreneurs who were eager to capitalize on it.

One entrepreneur in particular was especially aggressive in the manner in which he approached the DotNetNuke community. With significant exits under his belt from a number of previous business ventures, he had the connections and proven track record of being able to take an emerging opportunity to market. Without prior introduction, he called me on the telephone one afternoon, and we had a casual conversation about the state of the DotNetNuke project and some of the areas in which we were planning on offering professional services to the community. Later, at his request, we had a face-to-face meeting in Seattle near Pike's Place Market, where he told me some of his own business ideas and informed me that he would like to contribute to the financial health of the project so that it could achieve even greater growth. Offers such as this do not come without strings attached, so I warily kept my distance as I tried to learn more about his philosophy and values to determine if they were in alignment with the project.

Much to my surprise in March 2006, a press release was issued that announced that his company had raised \$1.75 million in seed venture capital based on a concept of providing an “economic platform” for developers leveraging the DotNetNuke framework. Clearly some of this so-called “economic platform” overlapped the professional services that we ourselves had begun to provide, and as a result the situation became complicated. At their insistence, a business meeting was scheduled in April at the Westin Towers in Seattle where some of their high-level business goals were presented to Scott Willhite and myself. Clearly the goal of the meeting was to convince us to make a commitment to formulating a deeper partnership. However, the meeting offered few tangible details on how our organizations were going to work together, other than the fact they wanted us to continue focusing on the technology while they focused on commerce.

It takes more than a few meetings or conversations to establish the trust

required to form a business partnership, and as a result we really did not feel comfortable moving forward at this juncture. The fact is, the DotNetNuke Board had been working very hard on the project for a number of years, and it was not clear to me how the members of our team were going to be included in the venture. In addition, some of the stories the entrepreneur had shared in an attempt to demonstrate his past business prowess had actually left an unpleasant taste in my mouth, as they appeared to not be aligned with the community ideals on which the DotNetNuke project was founded. When they realized that their open wallet was not going to result in open arms, I believe they were genuinely surprised and disappointed. However, this simply echoed the fact that they did not understand or share our philosophies or values. We advised them that they should first become contributing citizens to the community, establish a positive reputation, and then we could consider cultivating a deeper relationship. They agreed to act on this advice and so began a rather tenuous relationship in the months following.

I do think it is important to give credit where it is due. This serial entrepreneur saw the potential in DotNetNuke and was effective in painting a large vision for the project. He had some very solid business ideas that were based on his real-world experience in monetizing other technology platforms and industries. He was a skilled speaker, a tenacious salesman, buzz-word compliant, and knowledgeable on most of the *hot* technology and globalization concepts promoted through books such as *The World is Flat*, *The Tipping Point*, and *The Long Tail*. Like any good student, I absorbed as much of his wisdom as I could, and it really precipitated a change in my perspective from always looking at the project from a technical viewpoint to focusing more on the business model and broader ecosystem benefits.

The most important thing I realized from this experience was what a tremendous opportunity DotNetNuke represented and that we had reached a critical inflection point—if we did not take steps on our own to take the project to a higher level, somebody else would do it without our participation. Scott Willhite's wisdom and experience were instrumental throughout this process in terms of keeping us focused on the primary goals, which included building the DotNetNuke economy to its fullest potential, preserving the cultural roots of DotNetNuke and its universal accessibility, and rewarding those who have contributed (and continue to contribute) to its success.

Yin and Yang

In June 2006, I attended my first non-Microsoft technology conference, SYS-CON Enterprise Open Source in New York. I had been selected as a speaker, and my session focused on open source software on the Microsoft platform. Going in, I thought this was going to be hostile territory, but I soon realized that the “enterprise” focus resulted in the conference being technology-agnostic for the most part. The big news at the conference was that Marc Fleury, who was scheduled to do the keynote, was unable to attend because his company, JBoss, had just been acquired by Red Hat. SugarCRM, who had already completed a Series C round of financing, was a major sponsor of the event, and I spent a fair amount of time talking to its founders, recognizing the many similarities that existed between our platforms. Overall, the conference was a good experience and gave me a better sense of open source commercialization, especially in regard to high-tech start-ups and venture financing.

In the summer of 2006, we had our third annual board meeting, and one of most significant themes at the meeting was the concept of “balance.” In the past, we had always taken what we thought was an objective stance on the separation between the open source project and its commercial ecosystem. Because we were the stewards of the core project, we tried to avoid anything that could lead to potential conflict-of-interest scenarios. Generally this involved focusing on the open source community and avoiding direct interaction with commercial stakeholders. Interestingly, the commercial ecosystem seemed to thrive almost in spite of the fact that we were trying to ignore it. Gradually, we came to the realization that there were actually two very powerful influences in the project and that both were essential to its long-term stability—the “yin” and “yang” of DotNetNuke. These complementary forces needed to be embraced in order to preserve the delicate balance within the project and ensure its future.

Up until this point the DotNetNuke board had been serving the project in an unofficial capacity for a number of years, dealing with the various management tasks as best it could. Other than myself, the other members of the board either were self-employed entrepreneurs or were employed by other companies, which made it difficult to function as a cohesive team. Dan Caron had stepped down from the board in December 2005 due to the time commitment and amount of strain it was putting on his family. This left Scott

Willhite, Joe Brinkman, Nik Kalyani, and myself remaining as board members. As the months went by it became apparent that the project needed a different corporate structure and a full-time management team, but in order to support such an organization financially, it needed an adequate revenue base. And because the current services revenue was not scalable or predictable, it left little choice but to pursue alternative funding sources.

A New Company

It so happened that Scott Willhite had a good friend named Blair Garrou. Blair was managing director at DFJ Mercury, a seed and early-stage venture capital fund based in Houston, Texas. We had a conference call with Blair, and he indicated that although the DotNetNuke opportunity was not the right fit for his firm, that he would make some strategic introductions.

The first introduction he made was to Mark Radcliffe, a partner with DLA Piper who operates out of its Palo Alto office in California. Mark specializes in strategic intellectual property advice, private financing, corporate partnering, software licensing, and copyright and trademark matters. In the open source software realm, he is one of the most widely recognized and respected attorneys. After an initial meeting with Mark, we signed an engagement letter where DLA Piper agreed to defer billing for its services up to a certain threshold in exchange for a warrant to purchase stock in the company when it reached specific trigger conditions. DLA Piper was going to help us form an open source company that could better manage the needs of the DotNetNuke community and provide a solid business foundation for future growth.

DotNetNuke Corporation was formed September 21, 2006. Rather than coming up with a brand-new company name, we took the simpler approach, which had the benefit of providing a direct link between the open source project and the company (see [Figure 1.13](#)). The purpose of the new company was to assume a stewardship role and provide infrastructure, management, and support to the open source project as part of its regular operations. The previous board members, Scott Willhite, Joe Brinkman, and Nik Kalyani, all came aboard as official cofounders in the new entity. A commitment was made to transfer all of the existing intellectual property from Perpetual Motion Interactive Systems Inc. to DotNetNuke Corporation. A public press release was issued, and great care was taken to educate the community about the structural change to the project. Although there was some initial concern raised in regard to the “Corporation” branding, the community was overwhelmingly receptive to the change, and the transition created no serious disruption to the ecosystem. At this time the number of registered users on the dotnetnuke.com website was 335,000 members, and 2 million downloads had been recorded all time.



Figure 1.13

DotNetNuke's first challenge was constructing a business plan that would provide the foundation enough to sustain the project long term. The Benefactor program had been successful in providing members of the ecosystem with an opportunity to support the project and receive some additional benefits. Unfortunately, the number of participants in the program was not enough to generate sufficient revenue. In addition, we realized that the benefits being offered did not meet the needs of all community members. Specifically, there was a group of serious users of the platform who were in need of more professional support services, which were not offered through the program.

The week following the public announcement of the formation of DotNetNuke Corporation came news of another DotNetNuke event. The company that had its eyes set on creating an “economic platform” for DotNetNuke was hosting a private mini-conference in Las Vegas, Nevada. It had approached the majority of commercial vendors in the DotNetNuke ecosystem and had offered to pay for their expenses to attend the event. This

turned out to be a self-serving effort, as the main goal was to demonstrate and collect feedback in regard to module licensing opportunities. Ironically, even though the entire event was predicated on modular software leveraging the DotNetNuke platform, DotNetNuke Corporation had not been invited. This sent a clear message that we were not working together, and the unfortunate side effect was that the commercial vendors were caught in the middle. This would prove to be a challenging situation in the coming months, as much time and effort were spent on cultivating relationships and preserving the integrity of the ecosystem.

Larry Augustin

As part of his high-profile practice in the Bay Area, Mark Radcliffe had access to an enviable list of influential personal contacts in the software and venture financing industry. One of the first individuals he introduced us to in the fall of 2006 was Larry Augustin. Larry Augustin is an angel investor and adviser to early-stage technology companies. A member of the group that coined the term “open source,” he had written and spoken extensively on the topic worldwide. In 1993 he founded VA Linux (now GeekNet, NASDAQ:LNIX), parent company of Slashdot and SourceForge, where he led the company through an IPO in 1999 and served as CEO until August 2002. He is currently the CEO of SugarCRM and has advised and served on the boards of directors of a number of commercial open source companies including Appcelerator, Fonality, Hyperic, Medsphere, Pentaho, and XenSource.

I flew down to San Francisco in November 2006 and met with Larry at DLA Piper's offices in Palo Alto. We had a great conversation about the DotNetNuke community, the commercial ecosystem for extensions, and open source business models. Given Larry's background in enterprise Linux, I was initially curious as to why he would be interested in an open source project on the Microsoft platform. However, I soon realized that as a veteran entrepreneur, Larry was interested in any software ecosystem where open source was being leveraged as a disruptive business advantage. Larry had relationships with the majority of top-tier venture capital firms in Silicon Valley and specifically with the general partners who were receptive to open source business models. This initial meeting provided the foundation for a mutually beneficial and productive relationship between Larry and DotNetNuke.

Performance

Based on the feedback from hosting providers participating in the Microsoft Windows Shared Hosting Accelerator program, scalability and performance became a high priority in the fall of 2006. After many discussions, Microsoft actually allocated one of its experts on ASP.NET and Windows Server performance to work with us on optimizing the DotNetNuke application for the shared hosting environment. Charles Nurse spent a week in Redmond working side-by-side with this expert in the Patterns & Practices testing lab to learn how to effectively simulate load and performance test the application.

The scalability of the DotNetNuke application improved dramatically over this time frame, and the end result was released as DotNetNuke 4.4 in November 2006 to an overwhelmingly appreciative community. From this point forward, we had a regression baseline that could be used to compare new versions of the product in order to determine if performance had been degraded by new enhancements to the platform. As part of this process we also learned that there were very few developers in the Microsoft ecosystem who truly understood the ASP.NET/IIS/Windows Server dependencies or constraints from a performance perspective. We shared a lot of great knowledge with the general Microsoft developer community, and DotNetNuke's reputation as an enterprise web platform was bolstered.

In December 2006, more than a year after DotNetNuke 4.0 and ASP.NET 2.0 were released, we made the decision to sunset the DotNetNuke 3.x product. DotNetNuke 3.x was based on ASP.NET 1.1, and we had seen interest in this legacy platform drop off to the point where it no longer made sense to continue actively maintaining a parallel code base. Determining how and when to drop support for legacy versions of the Microsoft platform would continue to be a difficult challenge on an ongoing basis for the DotNetNuke project.

DotNetNuke Marketplace

The extensibility model in DotNetNuke had spawned a very active commercial ecosystem. By the end of 2006, hundreds of commercial modules and skins were available for the DotNetNuke platform. In addition, many companies were providing business services exclusively to the DotNetNuke market. This dynamic ecosystem was helping propel the growth of the project, but it was not without its share of issues.

Early in the project's history, a third party created a reseller environment that allowed developers and designers to sell their DotNetNuke products to consumers. This made it extremely easy for anyone, from a hobbyist developer to a serious independent software vendor, to get involved in the DotNetNuke commercial ecosystem. In the early stages, the existence of an established business environment for commercial components was critical to the growth of the project and adoption by business users. However, one of the most common types of feedback that we overheard related to this environment was about the questionable quality of third-party products and services.

Based on the reseller environment's low barrier of entry, the quality of commercial DotNetNuke components was extremely inconsistent. Some vendors were providing high-quality components, with professional support and explicit licensing terms. Others were essentially providing basic HTML scripts at a minimal fee with no support or licensing considerations. The combination of these polar opposites posed issues in terms of our goals to promote DotNetNuke as a professional framework. Effectively, the existing reseller environment was promoting a "Buyer Beware" mentality that was not complementary with our goals for taking the project to a higher level of business acceptance. In fact, some of the more serious independent software vendors told us that in order for them to get involved in the ecosystem, a more professional reseller channel would need to be made available.

To deal with the quality issue, we believed that a product review service could solve a number of problems. Although a comprehensive product review could provide great value, it would not be cost effective to perform, and therefore the review criteria would need to focus on some of the more fundamental product attributes such as whether the product installs and uninstalls properly. Although minimalist, such a review program would still provide value to the ecosystem. First, it would provide consumers with confidence

that the product they are purchasing is fully functional. It also would provide educational guidance to software vendors in terms of project standards and expectations.

We had approached the current reseller a number of times in the past with hopes that we could form a business partnership. The main benefit was if the reseller became a contributing citizen of the DotNetNuke community, we could work together to elevate the ecosystem to a higher level. It would also provide us with critical business intelligence related to the usage of the product. For us to effectively manage the product roadmap, it was becoming increasingly more important that we get in touch with our entire user community. The discussion forums represented a small but vocal group of community members who offered feedback, but there was a much larger group of users with whom we had absolutely no contact. Unfortunately, the reseller was not interested in working with us in this capacity, which left us with a single alternative: establishing our own reseller channel.

Combining the concepts of the review program with a reseller channel seemed to be a great way to satisfy a variety of project goals. Initially our reseller channel would only sell components that passed our review program. This would improve the overall perception of quality and confidence in the community and provide a new revenue stream to help us secure more dedicated project resources.

The development process of the reseller channel took longer than expected. In reviewing the requirements we recognized that there were no products with e-commerce functionality within the DotNetNuke ecosystem that could satisfy our needs. Therefore, we had to look elsewhere, and we were pleased to work out an agreement with AspDotNetStoreFront, an established product vendor providing a robust e-commerce solution to the Microsoft market. AspDotNetStoreFront was even interested in migrating a version of its software to the DotNetNuke platform, so we forged an agreement with them in hope of establishing a long-term business relationship.

The DotNetNuke Marketplace was launched in January 2007 (see [Figure 1.14](#)). Similar to dotnetnuke.com it took a minimalist approach to the website design. And one of the design goals was to ensure a consistent, professional user experience, as we felt it would be a good differentiator from the other reseller site. The number of products available grew slowly as awareness of the Marketplace grew in the vendor community, and product reviews were completed.

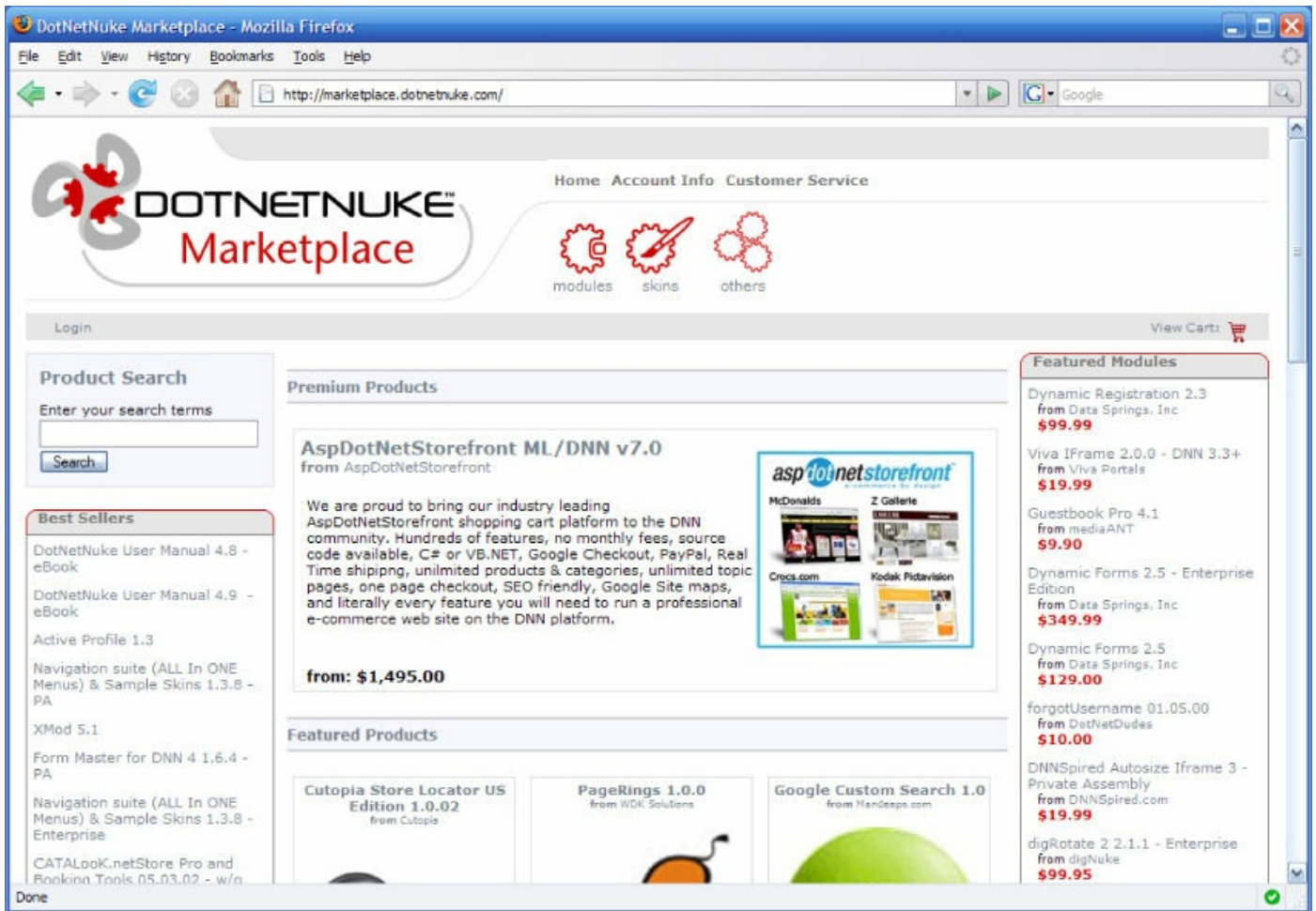


Figure 1.14

It did not take long before we learned a variety of valuable lessons. First, we had underestimated the first-mover advantage that the incumbent reseller had in our ecosystem. Without an incentive there was no motivating factor to encourage vendors to list their products in our marketplace. Without products, there was also no incentive for consumers to browse our marketplace and make purchases. The review program that we had assumed would be a great benefit actually became a barrier to entry, as we discovered that most vendors were not keen on paying a fee, no matter how minimal, to have their product reviewed. In hindsight this made perfect sense, because unless consumers are specifically demanding reviewed products, there is no motivation for vendors to invest in the program. In addition, our initial process for listing products was cumbersome, especially in comparison to the incumbent reseller. This resulted in some hesitation on the part of vendors to list their products and keep them regularly updated.

Another mistake we made was to maintain too much parity with the incumbent reseller in terms of the features and business model. In order to

be truly competitive, we needed to introduce some disruptive concepts and differentiate ourselves. As we learned these valuable lessons we adapted the Marketplace and slowly began to garner a greater inventory of products and consumer traffic, but still continued to lag behind the incumbent reseller. The most significant problem we faced was in regard to resources. Without start-up capital there was not enough revenue to allow for dedicated management of the Marketplace, and as a result it did not get the attention it deserved or required to achieve momentum.

Free Module Promotion

Without a working relationship with DotNetNuke Corporation, the funded entity that had promised to create an “economic platform” for DotNetNuke was aggressively trying to establish a foothold in the e-commerce portion of the ecosystem throughout the latter half of 2006. In July 2006, it had provided us with a proposal where it would pay us a royalty if we made it the exclusive e-commerce partner for the DotNetNuke ecosystem. However, based on the fact we were already establishing our own reseller marketplace coupled with the fact that we did not see eye-to-eye on many business practices, we decided not to move forward with this opportunity.

In December 2006, with participation from a number of commercial module vendors, it launched a promotion where a variety of the most popular modules were offered as a free subscription package for one year to DotNetNuke users. The package was mainly offered through hosting providers and community sites and was ultimately an attempt to build a large user base and validate their proprietary software licensing solution. Although the concept of a recurring revenue stream for commercial module vendors seemed attractive, it did not take long before a few critical problems came to the surface; the most significant was that commercial module vendors found themselves providing support services to users who had not paid for their products. This was not a viable model, and within a few months the majority of the module vendors pulled out of the promotion.

In April 2007 the company announced that it had received another \$5 million in venture capital funding and was expanding its vision to include commercial software outside the DotNetNuke ecosystem. We did not hear from them again, and by 2008 their focus appeared to have shifted away from software licensing and DotNetNuke as they worked on developing a website that provided product reviewers with tools to critique technology-based products and consumers the ability to browse the reviews.

Conferences

The first DotNetNuke conference occurred in May 2006 in Papendal, Netherlands. It was hosted by the Software Developer Network (SDN), which had recently added DotNetNuke as an officially recognized technology in its user group organization (based largely on the fact that Core Team member Leigh Pointer had successfully built a DotNetNuke user group of more than 300 members in the Netherlands). I was offered an all-expenses-paid trip to come and speak at the event, which I gratefully accepted. The DotNetNuke track ran in parallel with the other Software Developer Conference (SDC) tracks and featured sessions by Core Team member Vicenç Masanas as well as DotNetNuke experts from the Dutch community.

In October 2006, David Walker from Tulsa, Oklahoma, organized the first of what was to become an annual community technology event, which he named Tulsa TechFest. It was an ambitious conference, free to attendees, and funded by sponsors with many parallel tracks and notable speakers. Because he was a fan of the DotNetNuke platform, David reserved a track for DotNetNuke and invited me to come and speak. I had the honor of doing my first-ever conference keynote at this event, and it was a great experience to meet in person a number of Core Team members for the first time (John Mitchell, Chris Hammond, and Shawn Mehaffie) as well as a number of Microsoft MVPs.

Based on the rapid growth of the DotNetNuke ecosystem, by the end of 2006 we thought the community was ready for a dedicated DotNetNuke conference event. Conferences involve a significant amount of time, effort, and expertise to manage, so we recognized that our best approach would be to partner with an established conference organizer and potentially co-locate with an existing technology event. Initially we approached SYS-CON Media, but after a couple months of trying to work through contract logistics, we realized that some chemistry was missing from the relationship and that we would be better off looking for a different partner.

We approached a variety of other conference organizers and quickly learned that the majority of them either were already at capacity or were not willing to take a risk on a DotNetNuke conference event. Rather frustrated, we were at the point of giving up on the conference idea entirely when Joe Brinkman made contact with Shirley Brothers from DevConnections.

DevConnections is one of the longest-running independent developer

conferences focused on Microsoft technology and a perfect fit for the DotNetNuke platform. It turned out that Brian Goldfarb and Scott Guthrie from Microsoft had put in a good word for us with Shirley, and she was willing to entertain a DotNetNuke event that would be co-located with DevConnections at Mandalay Bay in Las Vegas in the fall of 2007. Working through the contract details with Shirley was a positive experience, and we ultimately agreed on two DotNetNuke conference tracks: one for developers and the other for designers and administrators.

One of the most significant benefits of co-locating with DevConnections was the fact that attendees were not restricted to only DotNetNuke sessions but were free to take advantage of any DevConnections content being presented in any track including ASP.NET, Visual Studio, and SQL Server. Nik Kalyani came up with a DotNetNuke conference brand of “OpenForce,” and Joe Brinkman took ownership of managing the conference logistics and spent considerable time in the months leading up to the event recruiting speakers, managing interactions with DevConnections, scheduling sessions, and leading marketing activities.

Based on the agreement struck with DevConnections, in the summer of 2007 we decided to approach the Software Developer Network in the Netherlands with a proposal of more officially partnering for its next SDC event, by using the “OpenForce” brand and assisting with marketing activities. An agreement was reached to provide two dedicated DotNetNuke tracks at the SDC event, and our first official European conference was scheduled for September 2007.

Microsoft Valuable Professionals

As the size and influence of the DotNetNuke ecosystem had grown, so had the visibility of its participants in the Microsoft community. Microsoft had a program called the Microsoft Valuable Professional (or MVP) program, which was designed to recognize individuals who made significant community contributions. Between the years 2005 and 2007, nearly 20 DotNetNuke Core Team members achieved this level of distinction.

In the spring of 2007, Microsoft held a global summit in Redmond for all MVPs worldwide, and this provided an excellent opportunity for our team to get together face-to-face. Because the majority of interaction between our geographically dispersed team occurs online, it was a great way to get to know one another and socialize. We booked a conference room at our temp office at Two Union Square and had team meetings during the week, including some entrepreneurial community members such as Lino Tadros from Falafel Software. We also treated the team to a group dinner at an Italian restaurant in downtown Seattle where the beer (micro-brew of course) and conversation flowed freely.

Fundraising

By the spring of 2007, DotNetNuke Corporation had prepared a business plan and was ready to present it to potential investors. Rather than leveraging friends and family, or even angel investors, we decided that professional or institutional investors should be our primary target. Nik Kalyani's past experience with fundraising was valuable in this process, and we felt confident that the product adoption and size of the community would be significant assets for our pitch.

Our relationship with Larry Augustin proved to be very valuable at this point, as he had many connections on Sand Hill Road. Larry was instrumental in setting up meetings for us with many of the top-tier venture firms in Silicon Valley. We met with Sequoia Capital, Accel Partners, Azure Capital, O'Reilly Alpha Tech Ventures, New Enterprise Associates, and Draper Fisher Jurvetson (DFJ).

Unfortunately, none of the firms we met with was interested in committing to DotNetNuke at this time. In a number of cases we had repeat meetings with the same VC, progressing from an initial meeting with an associate, to a general partner, and then to an all-partners Monday meeting. In general, the meetings were always positive; the VCs were courteous, thoughtful, and more than willing to provide advice on how we should capitalize on the opportunity. However, the most common piece of feedback was that we had not yet demonstrated a financial model that would create a “large company” opportunity. When VCs say “large company” they mean a company that can potentially reach a valuation of \$100 million dollars in five years. The frustrating part was that if we had already proven the financial model, there would be no need for investment capital at all. Other feedback included a preference for us to have a presence in the Bay Area because VCs prefer to have their portfolio companies nearby. And there was also some question about level of business leadership experience in the company. All of these items would need to be addressed one way or another in order for us to be successful in our fundraising efforts.

Awards and Accolades

The summer of 2007 was significant for DotNetNuke as the project received some recognition from a number of notable third parties. *Visual Studio Magazine* selected DotNetNuke as its Editor's Choice Winner for 2007, an award that had previously been given to Microsoft SharePoint in 2006. A month later, Info-Tech, an independent research group, selected DotNetNuke as a Leader in its Decision Diamond for Web Content Management for the small enterprise. The Info-Tech Decision Diamond award recognizes vendors that provide products and services of outstanding quality, with a strong enterprise strategy and high levels of customer satisfaction and retention. Both of these awards were unexpected and highly appreciated.

Later in the summer, we issued a release of DotNetNuke that contained support for the new OpenID authentication system. Because we were one of the first open source projects to implement OpenID, we received a cash award bounty of \$5,000. Dick Hardt, CEO of SXIP and a legend in the open source Perl community based on his tenure at ActiveState, was able to gain us passes to OSCON in Portland, Oregon, where the bounty award was presented. Other recipients of the bounty included Drupal and Plone.

Based on the publicity provided by the OpenID bounty, we were contacted by Microsoft with a proposal to integrate Windows LiveID support into the platform. This seemed to be a good fit, as many developers in the Microsoft world have become comfortable with the LiveID single-sign-on system. Integration also provided a sponsorship opportunity with the Microsoft Live team, a relationship that would reap rewards in the future.

DotNetNuke OpenForce 07

In September 2007, we had our first official European DotNetNuke conference in Papendal. The conference was branded DotNetNuke OpenForce Europe and had 80 registered attendees, each paying about €700. The conference attracted users from the United Kingdom, Ireland, France, Belgium, Spain, Portugal, Switzerland, Germany, Italy, Austria, Sweden, Norway, Denmark, and even from as far away as South Africa and Aruba. Joe Brinkman, Charles Nurse, and I attended the event from DotNetNuke Corporation and a number of Core Team members and Project Leads led sessions as well. The conference was a great success and established a solid working relationship with the Software Development Network for future events.

In November 2007, we had our first official North American DotNetNuke conference, co-located as planned with DevConnections at the Mandalay Bay Hotel and Casino in Las Vegas, Nevada. The conference managed to attract 225 registered attendees at \$1,500 and had two dedicated tracks spanning three days. Two vendors from the DotNetNuke ecosystem, Active Modules and R2Integrated, opted to become exhibitors in the DevConnections exhibit hall. The visibility that the project received at this event was incredible, as the DotNetNuke logo was displayed on all conference marketing materials alongside Microsoft products such as ASP.NET, Visual Studio, and SQL Server. Carl Franklin and Richard Campbell from the DotNetRocks! podcast helped host the final panel discussion for DotNetNuke Corporation, and at the conclusion the Core Team members in attendance finally got some of the public recognition they so generously deserved.

One of the ideas we had for OpenForce was to allow other Microsoft platform open source projects to participate at the event, and we were successful in attracting a number of distinguished guests including Scott Hanselman, Phil Haack, and Rob Connery. The ironic thing that happened, however, is that in the months leading up to the conference, each of these guests announced that he had accepted employment offers with Microsoft to work on the new ASP.NET MVC team. So when the conference finally arrived, we had an open source panel discussion, but the independent nature of it had definitely lost some of its impact. Regardless, we were still pleased to have been able to provide visibility and insight into a variety of open source projects on the Microsoft platform.

At the keynote for OpenForce North America we announced “Cambrian” as the new marketing codename for DotNetNuke 5.0. Nik Kalyani had come up with the name in reference to the “Cambrian Explosion,” a period in the earth's evolution where there was a dramatic increase in more complex life forms. We announced a roadmap that included the major features we were planning on implementing in the coming year as well as a tentative release schedule. We also mentioned that we were actively pursuing funding opportunities and shared some details on the current business model for DotNetNuke Corporation. Overall, this conference had a very strong business influence and demonstrated the momentum the project had achieved in professional and enterprise environments.

SLA Program

Initially mentioned at OpenForce North America and later rolled out publicly in January 2008, the DotNetNuke SLA program was a professional support offering available on an annual subscription basis by DotNetNuke Corporation. The program was introduced in response to community demand for professional support services for the DotNetNuke product and DotNetNuke Corporation offered a Bronze and Silver level to cater to the needs of different customers. The fact that open source projects make their source code available to everybody does not make everybody an expert. The fact that the open source software is offered for free doesn't mean that commercial services are irrelevant. In fact, open source software requires a dependable commercial ecosystem and a reputable vendor that stands behind its software and provides consumer confidence.

The SLA program was successful in engaging a variety of large customers who were using DotNetNuke in mission-critical deployments. For a long time we had relied solely on users communicating their issues through online channels, and as the software became more complex, it became increasingly more difficult to reproduce issues occurring in the wild. The information received through diagnosing customer problems directly in enterprise environments was essential in identifying and solving deficiencies in the software and building a higher-quality product. At this point it definitely became obvious that a healthy balance between open source users and professional customers was the optimal mix to create a powerful platform. With the introduction of the program, the legacy Benefactor program was phased out (it had never proven itself to be a scalable revenue generator) with some customers migrating to the new Sponsorship program, which provided visibility and marketing benefits, while others moved to the SLA program.

Although the program was successful, we also learned some unanticipated valuable lessons. First, because the only benefit being offered through the program was technical support, there was no incentive to purchase a subscription unless you were experiencing a problem and were in need of immediate assistance. Because the DotNetNuke software was so mature and stable, many companies had no immediate support requirement; therefore, the lack of a sales trigger significantly reduced our opportunity to engage with customers in a meaningful way. Second, we discovered that the vendors within our own ecosystem became our largest source of competition. As soon

as we announced our SLA offering, a number of vendors published their own SLA offerings at a lower price, undercutting our program and, in at least one instance, making an attempt to discredit the reputation of DotNetNuke Corporation. Obviously, these vendors had no way to support the DotNetNuke product itself as they had no ability to affect changes in the source code, but it did not stop them from getting some traction, albeit at the expense of the platform.

More Fundraising

In the fall of 2007, we had reorganized the management team and appointed Nik Kalyani as CEO. The changes were made in order to better reflect the roles we were playing in the organization and address some internal issues regarding vision and business leadership. We all agreed that Nik should focus his time and effort almost exclusively on fundraising, and he even volunteered to move himself and his family to California as he felt that we would have a better probability of getting funded if we had a person “on the ground” in the Bay Area. Due to a number of personal reasons, Nik had to delay his move until January 2008, but once there, it became obvious that his presence in Silicon Valley was definitely going to reap rewards for the company.

Following the success of the conferences, we made contact with a number of VCs (mostly via introductions from Larry Augustin). Firms we met with included Azure Capital and Benchmark Capital, both of whom allowed us to present to all of their general partners at an all-partners meeting. Although our pitch was much more refined by this time, we still had difficulty getting the VCs to believe in the revenue potential of the opportunity. One of the specific pieces of feedback was in regard to the “exit” potential. In a tech market where very few IPOs were occurring, the focus shifts to merger and acquisition (M&A) outcomes, and the VCs wanted to know who specifically would be interested in acquiring DotNetNuke in the future. This is obviously a hypothetical question, as nobody has a crystal ball, but we were still expected to have some solid defensible answers.

Another piece of feedback we received was that a “platform” was too broad of a category and was not focused enough to reveal a distinct monetization strategy. This led us to consider a more focused approach, and out of frustration we actually tried pitching an Enterprise 2.0 social networking application at one point to gauge the interest in it versus our platform approach. This was likely an unwise decision as the VC we pitched to was already familiar with our previous messaging and to shift gears so abruptly left them questioning the focus of our management team. Ultimately we reverted to the platform approach for future VC interactions, as regardless of the allergic reaction that some folks had, it was more closely aligned and fundamental to the success of the project to date.

In January 2008, a couple of significant events occurred. Larry Augustin ran

into an experienced entrepreneur named Navin Nagiah at a software conference. At the time, Navin was employed by Cignex, a successful systems integrator for open source content management systems including Alfresco, LifeRay, and Plone. Navin was looking for a start-up product opportunity with large potential, and Larry mentioned that he may want to take a look at DotNetNuke. Larry made the introductions via email and Nik followed up. In the same month, we received an anonymous solicitation through our website from Hummer Winblad Venture Partners, a respected VC from San Francisco. This was the first time a VC had come directly to us, and Nik worked very hard to close a deal with Hummer Winblad, doing many presentations and staying constantly engaged in the coming months.

When you get to the point of VCs wanting to call business references, you know you are getting close to a term sheet. Hummer Winblad asked if it could contact some of our references, and we supplied a number of enterprise customers and partners. This process was drawn out over a number of weeks, and we were even lucky enough to get some notable folks from Microsoft to speak with them about the DotNetNuke opportunity. In the end, Hummer Winblad's general partners could not get past their objection that Microsoft's volume pricing model makes it difficult for a company on the Microsoft platform to get larger enterprise sales deals. They indicated that greater sales potential was available on the J2EE platform, and it is these basic business principles, not the open source angle, which results in fewer investments in Microsoft platform vendors.

CodePlex

For a long time we had been looking for an organized way to allow community members to share their open source offerings with the community. Taking on the burden of managing the infrastructure for ad hoc projects was not something DotNetNuke Corporation could provide, as managing our own DNN projects had become enough of a challenge on their own.

In 2006, Microsoft had launched a new developer community site to replace the ill-fated GotDotNet. The new site was called CodePlex, and it provided a robust set of tools (based on the Microsoft Team Foundation Server architecture) that developers could use to manage their open source projects. The site was not solely available to Microsoft platform developers, but because of its close association to Microsoft, it soon developed into the go-to place for Microsoft platform open source projects. It provided some nice marketing capabilities, which provided projects with visibility and accessibility within the Microsoft ecosystem. And the fact that some teams from Microsoft Corporation were using CodePlex to manage and distribute their own out-of-band releases meant that there was commitment from the company to ensure the site was regularly maintained and innovated (a fundamental failing of its predecessor GotDotNet).

In late 2007, we approached Microsoft with a collaboration proposal where we offered to push our community members to CodePlex for managing its DotNetNuke open source projects. In exchange, CodePlex would contribute the infrastructure and also provide the DotNetNuke projects with some unique visibility to differentiate them from the other ASP.NET projects on the site. We worked closely with CodePlex team member Jonathan Wanagel on the integration, and in February 2008, the DotNetNuke Forge was announced.

In April 2008, there was another Microsoft Global MVP Summit in Seattle, which provided a great opportunity for the team to get together. We also leveraged the opportunity to invite a group of the more prominent vendors in the DotNetNuke ecosystem to participate in an information-gathering meeting to determine the most important attributes for a successful Partner program. Attendees included AppTheory, Seablick Consulting, T-Worx, Cybreze, Inspector-IT, R2Integrated, and Data Springs. Navin Nagiah attended the event to gain some familiarity and insight into the various

players in the DotNetNuke commercial ecosystem. And Jeff Loomans, a general partner from Sierra Ventures, a venture capital firm from Silicon Valley, flew up to Seattle to meet with us as well and discuss the DotNetNuke opportunity.

Security Issues

In the early spring of 2008, the project experienced a number of security issues that required our immediate attention as well as strategic management to ensure the reputation of the project was not tarnished. When it comes to security vulnerabilities in software, it is not always the technical issues that are the primary challenge but rather the motivations of the parties involved that play a significant role in defining an appropriate solution.

The first security issue was reported to us by Will Morgenweck of Active Modules, a well-known and respected vendor in our ecosystem. He indicated that his own site had been compromised, and he sent us his IIS logs in order to help us identify the problem. However, deep analysis of the logs and the application source code in the area targeted did not reveal the vulnerability. Without the ability to replicate the problem, it would be impossible to fix; therefore, we had to try to get to the bottom of it. When the third party had compromised Will's system, it had used a login account that provided some clue about its identity. I decided to take a chance and reach out to it via email; however, I was not confident that I would receive any response. Luckily, the third party did respond, and over the coming weeks I was able to establish a relationship through a series of email conversations.

It turned out we were dealing with a 22-year-old Iranian student named Morteza Kermani who was a member of the DotNetNuke Iran User Group. He indicated that he had not meant to cause any harm and would be willing to help us solve the problem. He explained the actions he had taken to bypass the security mechanisms, and this provided us with the detail we needed to replicate the problem locally. It turned out that he was relying on an undocumented behavior within the .NET Framework, which DotNetNuke had not taken into consideration. Basically, if a person specified a trailing period for a filename, the .NET Framework would not throw an Invalid Filename error but would instead strip the trailing period from the filename and then create the file on the disk. This vulnerability allowed Morteza to bypass DotNetNuke's file extension security, upload a shell script to the server, and then browse to it directly from a web browser, where he could then navigate the server file system. I would personally consider this .NET Framework behavior to be a bug; however, because we have no control over the underlying logic, we had to implement our own security mechanisms to prevent this type of exploit in the future. The patch was made available as

soon as we successfully validated our solution, and very few sites were affected.

The second security issue occurred in May and was much less severe in terms of the potential damage to the user's system; however, it was much worse in terms of public visibility. A group from Iran calling itself the ISCN, or Iran Security Center Networks, had discovered a vulnerability in the third-party FCKEditor rich text editor control that allowed an anonymous user to upload a file to a public website. The DotNetNuke file upload mechanism did have preventive code in place to prevent them from uploading malicious files; therefore, in most instances they simply uploaded a basic text file named ISCN.txt, which contained the following text:

```
!!! Persian Gulf For Ever !!!   Owned By : Magic-Boy , Imm02tal ,  
Mormoroth  
Contact Us : ISCNltd@GMail.com   ISCN Team   !!! Persian Gulf For  
Ever !!!
```

Although the text file did not represent a threat to a user's site, the ISCN group also posted links to every system it was able to successfully compromise on a security site called Zone-H. As the list grew, we knew we had to move very quickly to issue a patch or the reputation of the project as a secure platform would be affected. Tomotoshi Sugishita of the DotNetNuke Japan User Group and Mitchel Sellers were both extremely helpful in identifying and resolving the vulnerability.

The third security issue was discovered by a hosting provider within our ecosystem. In this case, the vulnerability was again not severe; however, it was the actions taken by the hosting provider that resulted in some serious problems. Rather than reporting the problem to our security alias and working with us to create a patch for the community, the hosting provider decided the security vulnerability represented a revenue opportunity for its business. It quickly created a “patch” support service that users could purchase to have the security problem immediately resolved on their sites. And then it issued a public press release on PRWEB announcing the existence of the vulnerability. This unprofessional behavior was not well received within the DotNetNuke developer community, and there was considerable backlash. Ultimately, the hosting provider did finally submit the problem to us and we were able to analyze its impact. In this case, the problem was related to manually invoking the install wizard, which could cause problems for some installations, as not all installation tasks are designed to be re-

executable. We were able to successfully resolve the problem almost immediately and issue a new general release.

IP Disputes

In April 2008, I received an unsolicited phone call from a person in San Francisco indicating that he owned the dnn.com domain name and was wondering if we were interested in acquiring it. Interestingly, the dnn.com domain name had previously been owned by media titan Knight Ridder Digital, and I had spoken to one of its attorneys in 2005 to determine if they were willing to part with the domain name but was told that they wanted to retain it. Somehow in early 2008, a domain name trader had managed to acquire the name from Knight Ridder, and he had decided to contact us because a vendor in the DotNetNuke ecosystem had expressed an interest in purchasing the name but had notified him that there may be trademark implications. Events transpired very quickly in the coming week, as the domain name trader tried to create a bidding war between ourselves and the vendor. Ultimately the price was too high, and we had to resort to legal means to try to acquire the domain name.

ICANN has a formal process known as the Uniform Domain-Name Dispute-Resolution Policy (UDRP). This policy is designed for situations that involve trademark-based domain-name disputes, typically where a complainant wants to acquire the domain name rights for one of its trademarks. The fact that we owned a trademark for the term “DNN,” coupled with the fact that the domain name trader had approached us and tried to extort a significant sum of money, led us to believe that we had a strong UDRP case. We hired an IP attorney and filed the necessary motions with ICANN.

The UDRP process is rather rigid, and what we discovered is that it tends to favor the domain name owner. It is up to the trademark holder to present a strong case in accordance with the UDRP criteria in order to try to convince a panel of judges that it should be the rightful owner to the domain in question. Demonstrating ownership of a trademark is fairly straightforward; however, demonstrating that an owner of a domain name is using it or plans to use it in bad faith to disparage your trademark is not easy. And this is not a legal proceeding; the decision is final—there is no provision for appeals.

In our situation, the domain name trader made a case that he was planning on using the domain name to create a custom website and had not had sufficient time to complete its construction. This seemed a bit far-fetched, given how eager he had been to try to sell the domain name to multiple parties. Unfortunately, the UDRP panel accepted this story and allowed the

domain name trader to retain ownership. Within a week of the decision he sold the domain name to the “Domain News Network” for an undisclosed sum of money. We were highly disappointed with the outcome and also learned a valuable lesson about the realities of the legal system.

Compounding our legal issues (and consuming our financial resources), in the summer of 2008 we received a notice from the United States Patent and Trademark Office (USPTO) informing us that a third party had filed a Notice of Opposition to our most recent application for the “DotNetNuke” trademark, as well as a Petition for Cancellation to the previously registered “DotNetNuke” trademark. The notices were filed by a hosting provider within our own ecosystem (the same one involved in the previous security vulnerability issue), and the basis of the complaints was that the DotNetNuke name was generic and “used in the computer industry to reference open source web content management systems.” This argument was flattering but far from reality, as DotNetNuke had clearly not reached the level of ubiquity where the term was being used in a generic way to describe various open source content management systems. In fact, DotNetNuke had never even been marketed as a CMS, but rather as a web application framework. Regardless of the frivolous nature of the dispute, as trademark owners we were required to defend ourselves or risk losing ownership of the mark entirely. The irony of this whole situation is that the freedoms we had provided to the community in regard to the use of our trademarks were now being used as a weapon by an individual against the community itself.

So again, we were forced into a complicated legal proceeding, a proceeding where the USPTO defined a schedule for submissions and disclosures that would take 13 months from start to finish. The only alternative to following the complete USPTO process was to come to a settlement agreement, and this was the solution recommended by our attorney. Through direct discussion with the hosting provider we realized that the biggest problem leading up to the legal filing was a lack of communication and understanding on the goals and motivations of each party. The hosting provider had built a business and was afraid of how changes in the trademark policy could potentially affect its livelihood. From our perspective this appeared rather paranoid, as this hoster was only one of many organizations that were conducting business in the DotNetNuke ecosystem, and we understood that ensuring the viability and longevity of all of these entities was definitely vital to the success of the project. Regardless, we were able to successfully structure an agreement that

dealt with their concerns and allowed us to avoid a lengthy and costly legal process.

Term Sheets

Throughout the late spring and summer of 2008, Nik Kalyani worked closely with Navin Nagiah to get him up to speed on the DotNetNuke ecosystem. Navin had already quit his job at Cignex, and he was eager to join the team; however, from a cash flow perspective, we were not able to accommodate his needs. We constructed an agreement where he would act as a business adviser to the company and commit 100 percent of his time to fundraising. In exchange, we agreed that he would come aboard as CEO post-funding.

Navin had worked in the Bay Area for quite some time and had his own network of trusted business advisers and associates that he was able to leverage for VC introductions. Navin got to work immediately, setting up meetings with investors and pitching the DotNetNuke opportunity. In the late spring and summer of 2008, Navin made contact with many reputable firms including Sigma Partners, El Dorado Ventures, Charles River Ventures, SAP Ventures, Walden International, Emergence Capital, Matrix Partners, Trinity Ventures, and Menlo Ventures. In most cases, Nik accompanied Navin for the in-person meetings, and I participated via conference call. Sometimes we were dismissed after an introductory conversation, and in other cases we did presentations in all-partner meetings. By this time our pitch was becoming very clear and consistent. But although the interest among these firms was high, there was still something holding them back from taking the next step. The biggest issue still seemed to be a lack of confidence in whether an open source company could reach critical mass on the Microsoft platform. And although we could provide metrics and indicators to help mitigate this risk, it ultimately came down to a gut reaction that left the VCs feeling uneasy.

By midsummer 2008, we had reduced our list of seriously interested firms down to three: Onset Ventures, August Capital, and Highland Capital Partners. In the case of Onset, we had met with it repeated times, with our introductory session occurring through Navin back in February 2008. Onset had been a great firm to work with through the process, as its team had provided a great deal of advice and guidance that helped us clarify our message as well as our market opportunity. Our interactions with August Capital were through general partner Vivek Mehra, whom we found to be very direct and insightful, and the fact that one of August Capital's co-founders, David Marquardt, had been on the Microsoft board of directors since 1981 was a definite plus. Navin flew to Boston to meet with Highland and had a

productive meeting, but because it was an East Coast firm, it made follow-up communications and in-person meetings with the partnership a bit more challenging. Although each of these firms showed significant interest, had met with us repeatedly, and had spoken to our business references; none of them was making a funding decision. We felt that we needed a catalyst of some sort to bring the process to a climax. That catalyst came in a most unlikely form.

At the MVP Global Summit in the spring I had promised Oliver Nguyen that I would do a DotNetNuke presentation at his BAY.NET User Group the next time he had an opening for a speaker. The user group meeting was scheduled for August 27, and it provided a great opportunity to give the investors some real-world exposure to the DotNetNuke project and community. The event attracted about 50 members, and general partners from August Capital, Onset Ventures, and Highland Capital were all in attendance. This was one of the most nerve-wracking presentations I have ever done, and I was very relieved to be able to pull it off without a hitch. Nik Kalyani provided me with support during the Q&A section, and Oliver Nguyen and the BAY.NET User Group leadership were great hosts of the event. It turned out that this meeting created the additional motivation we were looking for, as two of the investors decided they wanted to move forward, and we received two competing term sheets. And thus began the real education when it comes to venture capital funding.

All things being equal, the term sheets we received were similar in a number of ways. The pre-money valuation (the current value of the company) and the investment amount offered were identical, as was the amount of equity in the company the investors were demanding for themselves and the portion of equity they wanted to carve out for an options pool. Both term sheets were also based on syndicated deals, where the investor needed another VC partner to come aboard to complete the deal (unfortunately, however, the two firms did not want to work together or else it would have potentially made the entire process much easier). This is how a VC reduces risk in an early-stage investment, but it definitely added a dilemma for us, as signing a term sheet with only a 50 percent commitment does not guarantee that you will find a partner for the other 50 percent. Items that differed in the term sheets were that one firm was offering funding in “tranches,” basically meaning that the investment amount would be provided in multiple installment payments when specific milestones had been reached. This “tranching” approach

coincided with their opinion that we were missing some key business leadership in the company, which meant that we would immediately have to perform an executive search to bring in a heavy hitter. And where one term sheet had a “no shop” clause preventing us from shopping around for better terms, the other term sheet had no restriction in this area.

After much deliberation (and day and night conference calls), we decided to accept the term sheet from August Capital near midnight on September 2, 2008 (the other term sheet was scheduled to expire on September 3). The reasons for this decision were that we felt most comfortable with the style and approach of Vivek Mehra, the general partner on the deal; the reputation and pedigree of August Capital would ensure higher-quality advice and strategic opportunities; the chemistry of the current management team could be maintained; and we could focus immediately on executing the business plan rather than performing an executive search; and given the uncertainty of the global economy, we wanted to get the entire investment amount into our bank account in one lump sum. In addition, this term sheet did not have a no-shop clause and would have provided some flexibility if things went sideways.

Although we had accepted a term sheet, it did not mean we had completed the funding process. We still needed to find a partner to join August Capital on the deal. Our earlier relationship with Sierra Ventures from the Global MVP Summit became advantageous at this point, because based on its previous interest in the opportunity, coupled with our signed term sheet with August Capital, Sierra invited us to an all-partners meeting in the morning on Monday, September 8. At the conclusion of the meeting, they asked us to stick around, and within half an hour we received confirmation that they wanted to become the syndicate partner on the deal. The next decision we had regarded which partner from Sierra would manage the investment, as we had previously engaged with two members of their team, Jeff Loomans and Tim Guleri. We went out for a celebratory dinner with August Capital and Sierra Ventures the next evening, and afterward, based on Tim's more extensive open source experience with commercial open source start-up, SourceFire, we concluded that he would be the better candidate.

After signing the final version of the term sheet on September 11, we moved on to the due diligence stage. Due diligence involves the disclosure of every legal contract or agreement that has a bearing on the company's assets or liabilities. Essentially, the investors are acquiring partial ownership of the company and need to ensure that everything is in order from a financial and

legal perspective. Because of the maturity and complexity of the DotNetNuke open source project, the due diligence process in our situation was more complicated than average. We needed to dig up executed copies of all Contributor License Agreements, Software Grants, Non-Disclosure Agreements, third-party consulting contracts, sponsorship agreements, advertising agreements, independent contractor agreements, trademark registrations, domain registrations, financial records, tax returns, incorporation documents, and so on. At one point I realized that I had spent three full weeks doing nothing but collecting paperwork, signing it, faxing it to our attorney, and then couriering the physical documents.

DotNetNuke OpenForce 08

As is customary for the month of June every year, Microsoft was hosting the TechEd conference in Orlando, Florida. However, for the first time in 2008, Microsoft decided to split the Developer and IT Pro tracks into two consecutive weeks. This left the convention center empty over the weekend between the two weeks, and Microsoft graciously made the space available to community groups.

A couple of eager members from the DotNetNuke community in Florida, Brian Scarbeau and Michael Webb, convinced Joe Healy from Microsoft that DotNetNuke could leverage a room for its own developer event. With the assistance of DotNetNuke Corporation and especially from advertising manager Bill Walker, OpenForce Connect became a reality. A large contingent of vendors from the DotNetNuke ecosystem stepped forward to sponsor the event, and a variety of prizes were donated to be distributed among attendees. Overall, the mini-conference was a great success, and the thing I found most interesting was the fact that many attendees had traveled long distances to attend OpenForce Connect, even though they had no intentions of attending TechEd.

In October 2008, we had our second annual DotNetNuke OpenForce Europe conference in the Netherlands. Co-located with the SDC, this time the event was moved to Noordwijkerhout, which is located near Amsterdam. The overall attendance to this conference was down slightly from the previous year, but this was not surprising given the current state of the global economy. We had two tracks spanning two days, and the conference once again provided a great opportunity to network with members of the European community.

November 10–14 we had our second annual DotNetNuke OpenForce North America conference (see [Figure 1.15](#)), once again co-located with DevConnections at Mandalay Bay in Las Vegas, Nevada. We had two tracks spanning three days, and we added a DotNetNuke training day as well that was hosted by our official training partner, Engage Software. Overall attendance to the conference was down, but the number of vendors who participated in the exhibitor area increased dramatically. In addition to DotNetNuke Corporation, there was representation from Active Modules, Data Springs, IowaComputerGurus, Seablick Consulting, R2Integrated, AppTheory, Engage Software, iHOSTASP.net, and PowerDNN. This

participation definitely increased the visibility and impact of DotNetNuke at the conference over the previous year, as I overheard more than one conference attendee proclaim “DotNetNuke is everywhere this year!”

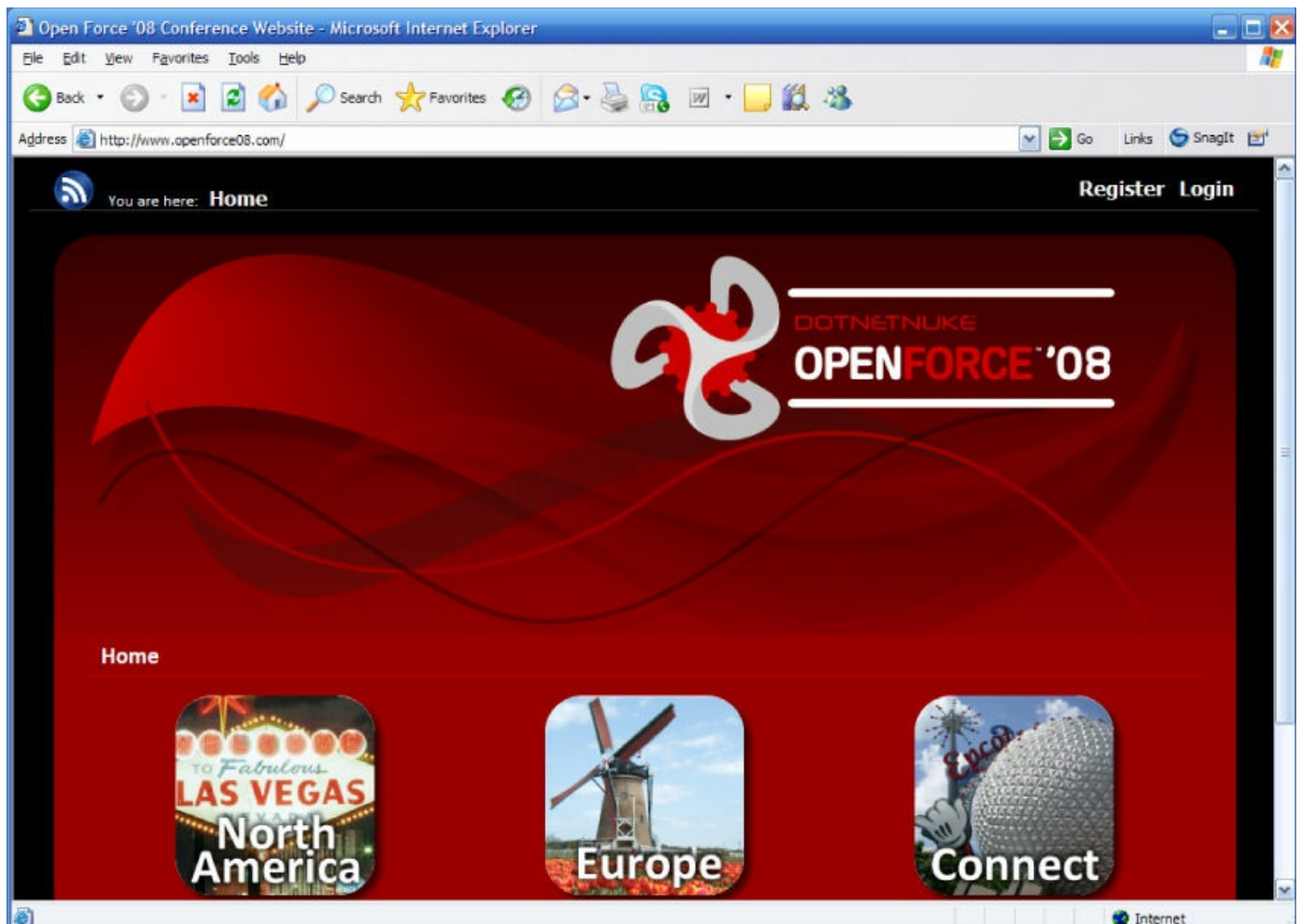


Figure 1.15

Bill Walker worked closely with Will Morgenweck to schedule a community event one evening where attendees of OpenForce could get together and socialize in a casual setting, and people would be eligible for prizes donated by vendors. With more than \$80,000 in prizes up for grabs, it actually convinced a number of DevConnections conference attendees to switch their registration to the OpenForce track so they could attend the community night. In addition to the community night, R2Integrated also created a social networking site at dnnconnections.com, which transmitted live podcasts, news, and interviews from the conference so that the community could feel more connected.

We had hoped to make a funding announcement at the conference so that it would have the greatest impact; however, the due diligence took longer than

expected, and we missed our window by a couple weeks. Although we could not mention the imminent investment, we did take the opportunity to make another major announcement during the keynote.

DotNetNuke Professional

As we worked through our business plan, we had looked at a variety of business models that other open source companies were using to successfully balance the requirements of commerce and community. In many cases, companies were making a commercial version of their open source product available under a commercial license with a yearly subscription model. The commercial version provided access to expert technical support and value-added network services that simplified and optimized the development and maintenance of the product. In fact, serious business users of the DotNetNuke platform had been demanding this for quite some time. In some ways, it was simply a repackaging of our existing SLA program, but in others it was a completely new strategy and direction for the project, including a new focus on positioning ourselves as a content management system (CMS).

DotNetNuke Professional Edition was announced at the OpenForce North America conference and was promised to be available in Q1 2009 (see [Figure 1.16](#)). It would be based on the mature DotNetNuke 4.9 code base and would include the essential modules for building a robust site. In keeping with the spirit of the open source development model, we promised to work continually to provide new innovation and increased value in the free, open source core product, which would also benefit customers of the commercial edition.

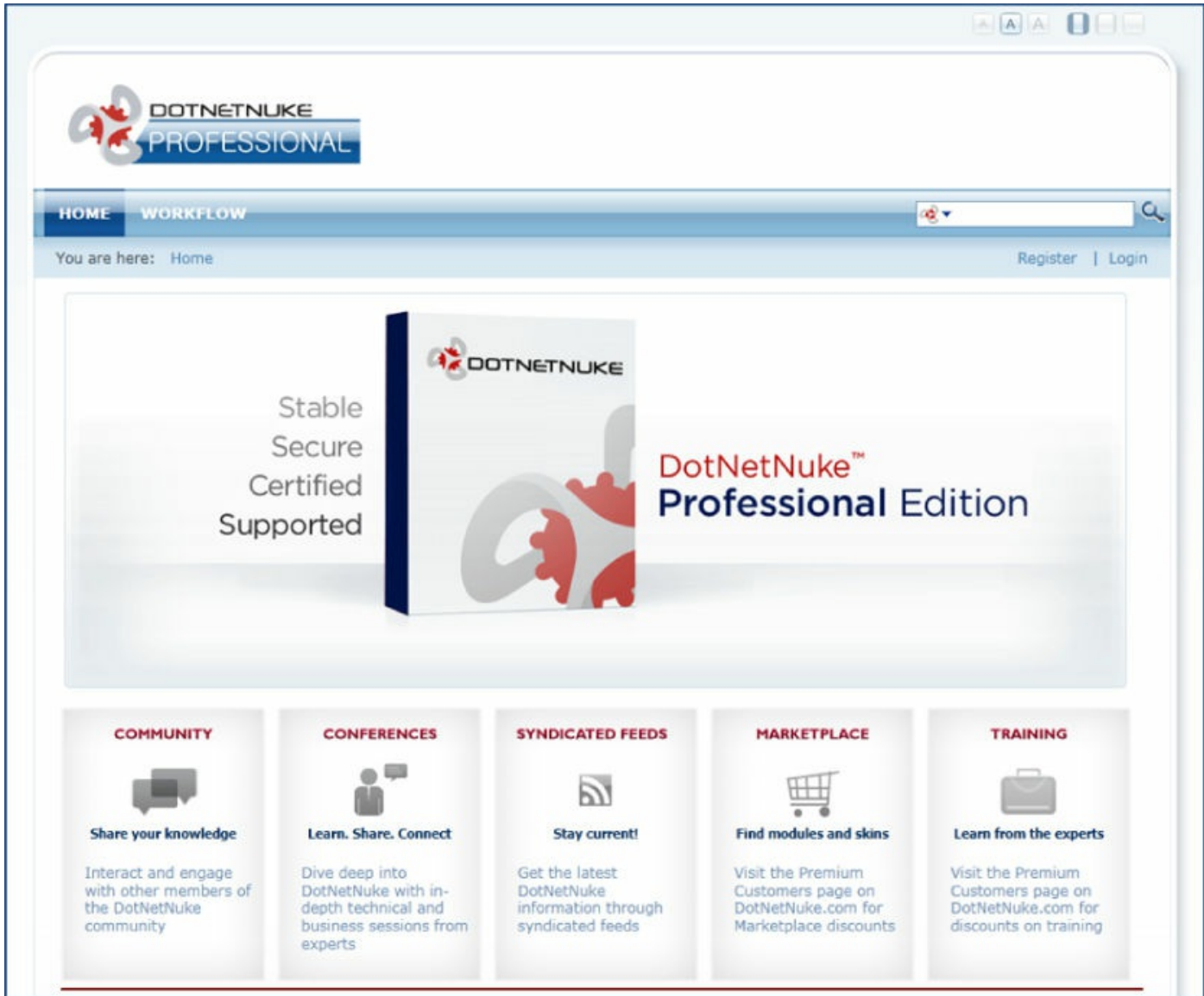
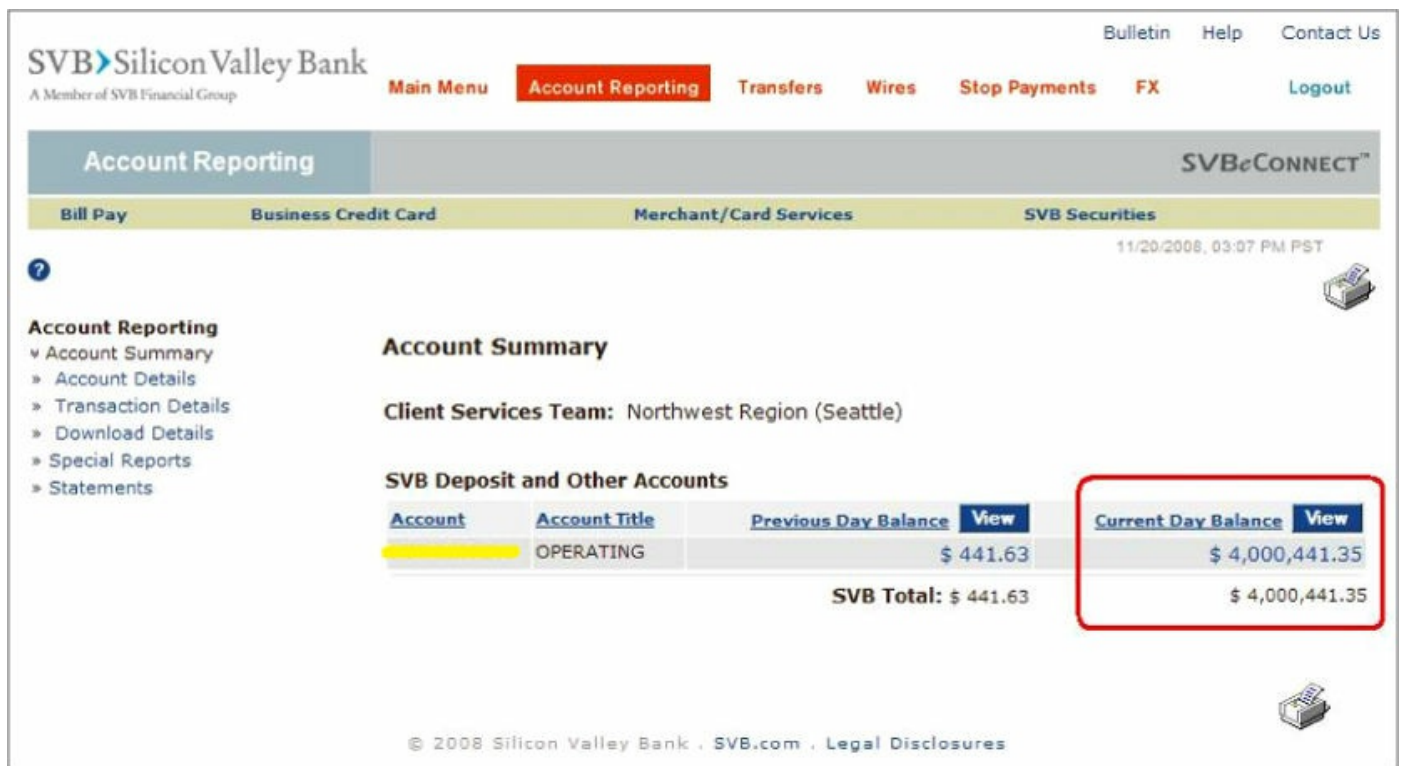


Figure 1.16

Series A Announcement

The due diligence ended up taking 10 weeks to complete (it usually takes 4–6 weeks) but also served a useful purpose in terms of getting all of the legal artifacts within the company in order. Aside from the internal due diligence, there was also the funding documentation itself as well as the related filings to provide preferred shares to the investors. Our legal team at DLA Piper worked very hard to ensure that all bases were covered, and we successfully closed the deal. The actual funding hit our bank account on November 20, 2008 (see [Figure 1.17](#)), and we made our public Series A announcement on November 25, 2008 (see [Figure 1.18](#)).



[Figure 1.17](#)

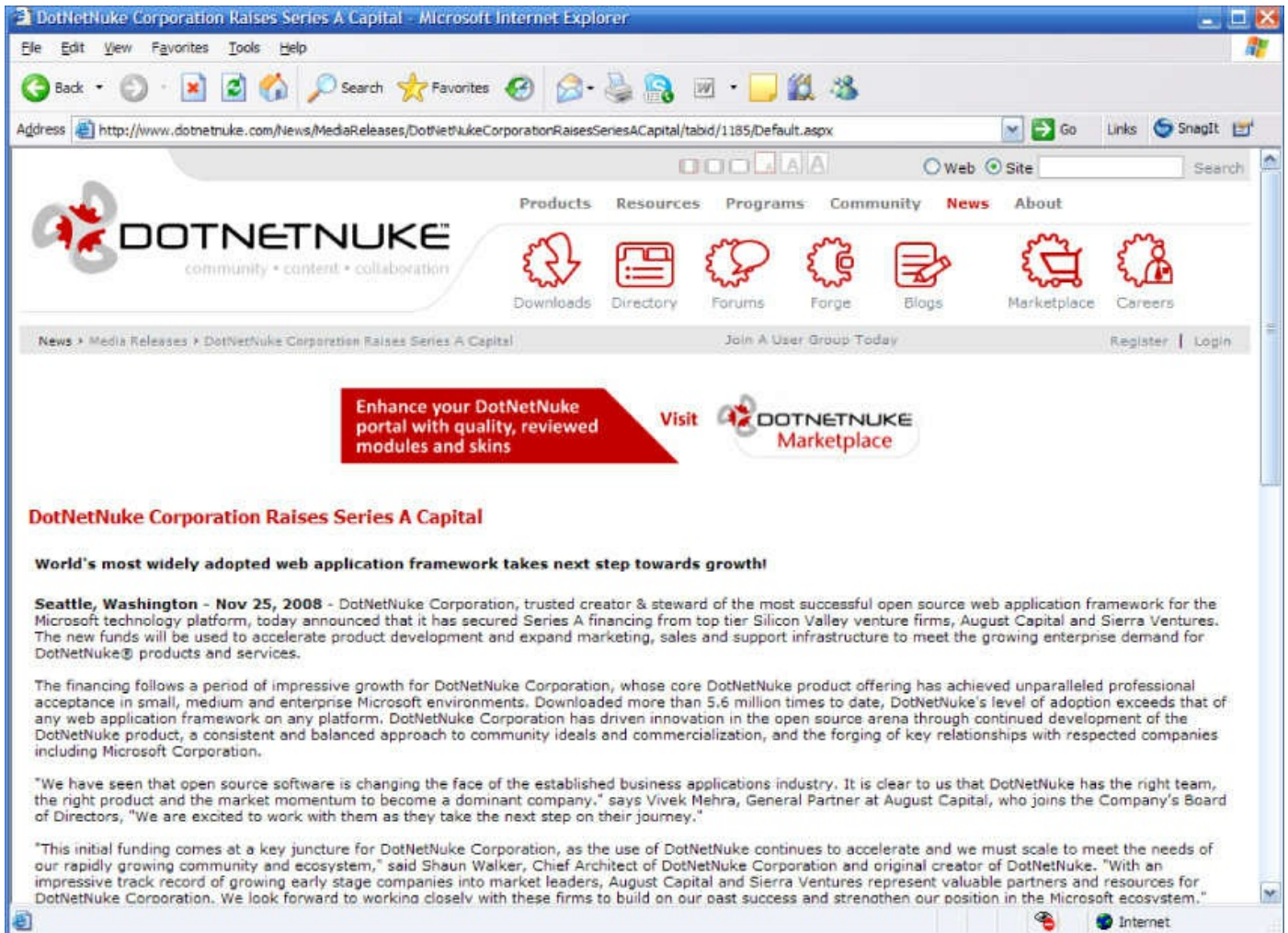


Figure 1.18

The following week, we announced that Navin Nagiah was officially joining the company as CEO. It had been a long journey for Navin, as he had originally been introduced to us in January 2008 and had worked full time with us for four months without any compensation as we tried to close our round of funding. Navin had certainly demonstrated his commitment and faith in the opportunity, and we were glad to have him come aboard. There was a lot of work to be done on an aggressive schedule, but we finally felt like we had all the pieces of the puzzle in place to start the next step of our journey.

The new board of directors consisted of Navin, myself, Vivek Mehra from August Capital, and Tim Guleri from Sierra Ventures. We needed one more external party to join the board of directors, and Larry Augustin was the obvious candidate. Because Larry was already involved in so many other high-profile, funded open source companies, it took some arm twisting to convince him to come aboard. But ultimately he agreed to join the team, and a public

press release soon followed.

Physical Offices

For the prior three years DotNetNuke Corporation had been operating as a purely virtual company, with all employees working remotely from their home offices. With funding completed, we were finally able to set up some actual brick-and-mortar headquarters. Based on the fact that Charles Nurse and myself were based in British Columbia, Canada, we decided to establish the technical engineering office near Vancouver. We chose Abbotsford, a city that is one hour east of Vancouver in the Fraser Valley, as that was the city where I currently lived (and it was only a 30-minute drive for Charles).

I secured a lease in an old office building that offered us about 1,000 square feet of usable space. We had a local contractor do some renovations to make it more applicable to a start-up software company (the original space had been used as a retail space for selling bridal gowns, and, in fact, my office had previously been used as a fitting room). Based on its age, the office building had a lot of “character,” and some of the original employees will fondly remember the uneven flooring, which was spongy to walk on; the bathrooms with unreliable pipes and no heating; the wooden deck at the back, which was so rotted that I only ever saw one brave soul attempt to stand on it; and the family of squirrels that lived inside the ceiling and occasionally dropped in through a hole for a brief visit. Some of the first employees hired during this first year were Sarah Darkis, John Lucarino, Israel Martinez, Rob Chartier, Candice Whyte, and Ken Grierson. Within a year we quickly outgrew the space in Abbotsford and moved to a more modern office building in Langley, British Columbia, in Walnut Grove near the intersection of the #1 freeway and 200th street exit.

At the same time that the office was established in Abbotsford, we secured some co-located space in a building in San Jose where one of August Capital's portfolio companies, a start-up company named Sky Pilot, had extra capacity. This space served our needs until we secured a more permanent space of our own in San Mateo, in the Crossroads Towers off near the intersection of the 101 and 92 freeway, near [SalesForce.com](https://www.salesforce.com)'s headquarters.

We chose California as our business headquarters for a couple of reasons. Navin Nagiah and co-Founder Nik Kalyani were both living in the Bay Area, and there was a lot of sales and marketing talent available to help us establish our business presence. And the fact that both August Capital and Sierra Ventures were based in Palo Alto made it easy to accommodate board

meetings and other interactions with our investors.

DotNetNuke 5.0

After the initial announcement of Cambrian at OpenForce North America 07, not much news had been shared with the community about its ongoing development. Our roadmap slipped behind schedule due to our focus on fundraising, and we made a number of releases to the 4.x product to deal with some security issues and improve the overall stability of the application based on insight gained through the SLA program.

Meanwhile, Charles Nurse continued to work diligently on DotNetNuke 5.0, and by the summer of 2008 we had reached a point where we believed we were code complete on all of the major enhancements that had been introduced to the platform in this iteration. We had not tackled all of the features promised in the Cambrian roadmap, but we had implemented a lot of fundamental changes that would be essential to delivering future functionality.

DotNetNuke 5.0 may not have appeared to be a significant release on the surface, but once you dug a little deeper, you quickly realized that there were a ton of major enhancements. The entire packaging format for extensions had been overhauled, security had been improved through Deny permissions and other refactoring, performance had been optimized with a brand-new data caching pattern, the administrative area had been opened up to allow for complete flexibility, page creation and management had been streamlined, the skinning engine received some designer-friendly new concepts, and, perhaps most importantly, the overall stability and quality of the application was maintained.

Keeping with tradition, DotNetNuke 5.0 was publicly released on December 24, 2008, six years from the date that IBuySpy Workshop had originally been released.

Day of DotNetNuke

In June 2009, we also had the first of many community organized events under the brand name of “Day of DotNetNuke.” Day of DotNetNuke was the inspiration of Will Strohl, a loyal DNN evangelist and supporter from Florida. The concept behind Day of DotNetNuke was that it was a free event, completely organized by community volunteers and funded through sponsorship, which usually took place on a Saturday. The event provided an educational opportunity for users to attend sessions related to the DotNetNuke platform, presented by volunteer community experts. And it also provided the ability for members of the ecosystem to network with one another and promote their products and services.

The first event occurred in Tampa, Florida, and was a huge success (largely because of Will's exceptional commitment and charisma in leading the event). This, of course, led to more Day of DotNetNuke events worldwide in the years to follow, including Chicago, Nova Scotia, Paris, Orlando, and Charlotte.

The event in Charlotte in the spring of 2013 was especially ambitious, organized by the Queen City DotNetNuke User Group (QCDUG) and using a unique theme of “Southern Fried DNN.” Clint Patterson, Allen Foster, Robb Bryn, Fred Ellise, and Ryan Moore did an amazing job of both evangelizing DotNetNuke as well as showing people Southern culture and hospitality.

DNN-Europe

DotNetNuke had always enjoyed a very loyal group of users in Europe. In fact, the original Core Team had a significant representation of European members, and they were very active in all of the community channels. Eventually, this group decided to establish a more formal organization, and Sebastian Leupold from Germany took the initiative in terms of creating a membership-based website at dnn-europe.net for the Network of DotNetNuke Professionals.

DNN-Europe provided a website for European users to congregate and collaborate on DotNetNuke issues and business opportunities. DNN-Europe was especially involved in the localization of the DotNetNuke application, as well as with the creation language packs so that users could utilize the administrative user interface in their native language. It also resulted in the creation of an annual retreat where members could get together in person. The retreats occurred in many different countries as different members stepped forward to act as hosts. DNN-Europe was also heavily involved in official DotNetNuke conferences in both Paris, France, and Hamburg, Germany.





DNN-Europe would later be replaced by DNN-Connect, an official nonprofit organization registered in Switzerland and founded by Peter Donker, Vicenç Masanas, and Philipp Becker.

Snowcovered Acquisition

In 2009, we were trying to make some decisions on what to do with the DotNetNuke Marketplace. We had invested a lot of time and energy into creating a marketplace for vendors selling commercial third-party extensions to the open source platform, but from a business perspective we could not convince vendors to switch from using the original DotNetNuke marketplace at Snowcovered.com.

We decided to reach out to the owner of Snowcovered, an individual named Brice Snow who lived in Paris, Tennessee. I made contact with him via email and introduced Navin who then flew to Paris to meet Brice in person. Brice was a very down-to-earth entrepreneur who had grown his marketplace from scratch into a very profitable business. However, as is sometimes the case with success, operating the marketplace had become a 24/7/365 responsibility for Brice, and it was beginning to take a toll on him. He had not been able to take a vacation in years, and he was even starting to feel like his health was being affected from the lack of sleep and exercise. So it turned out that Brice was quite interested in exploring the concept of selling the Snowcovered marketplace.

Navin worked with Brice to establish a suitable business arrangement, and in September 2009 we made the announcement that we had acquired Snowcovered (see [Figure 1.19](#)). Brice joined our board of advisers and committed to helping us with the transition. We shut down our own marketplace and focused our efforts on taking over the operations of Snowcovered. Snowcovered was a sophisticated e-commerce platform that was built on top of the DotNetNuke platform, and it took many months before we felt fully comfortable with it. We decided that it made sense to retain the Snowcovered brand following the acquisition so that there was no disruption in the ecosystem. The brand was maintained for a few additional years until we eventually did a rewrite of the e-commerce codebase so that we could upgrade a more modern DNN version and rebranded it as the DNN Store.


SHOP | **POST** |  Cart |  Help | *Not logged in* | [Login](#) | [Register](#) |  Promos
[View Open Requests](#) | [Technical Support](#) | [Customer Service](#)

Search


Browse


- [DotNetNuke 5](#) (2499)
- [DotNetNuke 4](#) (6426)
- [DotNetNuke 3](#) (2410)
- [DotNetNuke 2](#) (527)
- [DotNetNuke 1](#) (71)
- [SharePoint](#) (26)
- [Graphics & Images](#) (5)
- [Web Server](#) (5)
- [By Site Type](#) (25)
- [Windows](#) (17)

[Add Reviews](#)

Bargains

Modules Spotlight





[Bulk Emailer - Advanced DNN Email Module .48](#) by [Interactivewebs.com.au Australia](#)

\$99.00

Bulk Emailer for DNN 4 and 5 is an advanced Email module with some interesting features. Reliable and now with the ability to send directly from MS Word. ... [read more](#)


[DNN SilverLight Video Library 3.0](#) by [Interactivewebs.com.au Australia](#)


\$39.95

Play .wmv files directly in your DotNetNuke website with the most advanced Silverlight module available for DNN. ... [read more](#)

[See more new in Modules](#)

English

Most Popular in Modules 



1. [Data Springs Collection 3.0 \(23 Modules\)](#)
by [Data Springs, Inc.](#)
2. [Event Calendar and Registration 3.0](#)
by [Invenmanager.com](#)
3. [Ultra Media Gallery 5.5](#)
by [BizModules.net Solutions](#)

Figure 1.19

Telerik Partnership

A hot trend in 2009 was related to user experience, as developers and customers were increasingly looking for ways to utilize more client-side technology that produced a more responsive experience in their web applications.

We evaluated a number of options, and unfortunately at the time there were not many good open source UI frameworks that would allow us to accomplish our goals. So that meant we needed to consider commercial UI frameworks, and we met with the leading ASP.NET UI vendors at the time, including Infragistics and Telerik. Based on our observations at the time, it appeared that Telerik had the most comprehensive UI control suite for ASP.NET, and it was willing to work with us to establish a model where we could distribute its commercial controls with both our commercial and open source editions.

We announced a partnership with Telerik in September 2009, which allowed DNN developers to leverage the RAD Controls for ASP.NET Ajax. This provided a huge amount of value to folks in our ecosystem as they no longer needed to purchase developer licenses for these controls from Telerik. And it also provided benefit to us, as we were able to utilize these controls in our own development efforts to improve the overall user experience of the DNN application.

Series B

In early 2010, our commercial business results were looking very positive, as we had achieved our revenue goals every quarter since we launched our commercial product edition. One of our board members, Tim Guleri (from Sierra Ventures), was singing our praises to other venture capitalists, and a firm from Utah named UVP (Utah Venture Partners) became interested.

At this point we were managing our Series A investment very wisely and were not in a position where we needed to raise additional funds. However, given the global economic climate at that time, we knew that we should not ignore any opportunity to improve our financial position. So when UVP contacted us, we quickly put together an investor deck, and Navin and I flew to Salt Lake City to meet with the UVP partnership.

The meeting went well, and in sharp contrast to our earlier experiences with fund raising, progress moved very quickly, and we were soon entertaining another term sheet—this time offering a more substantial round of funding. The amount was in excess of \$8 million dollars and was comprised of investment from UVP as well as from August Capital and Sierra Ventures who both wanted to exercise their pro-rata rights to avoid dilution and retain their equity stake in DotNetNuke (see [Figure 1.20](#)).

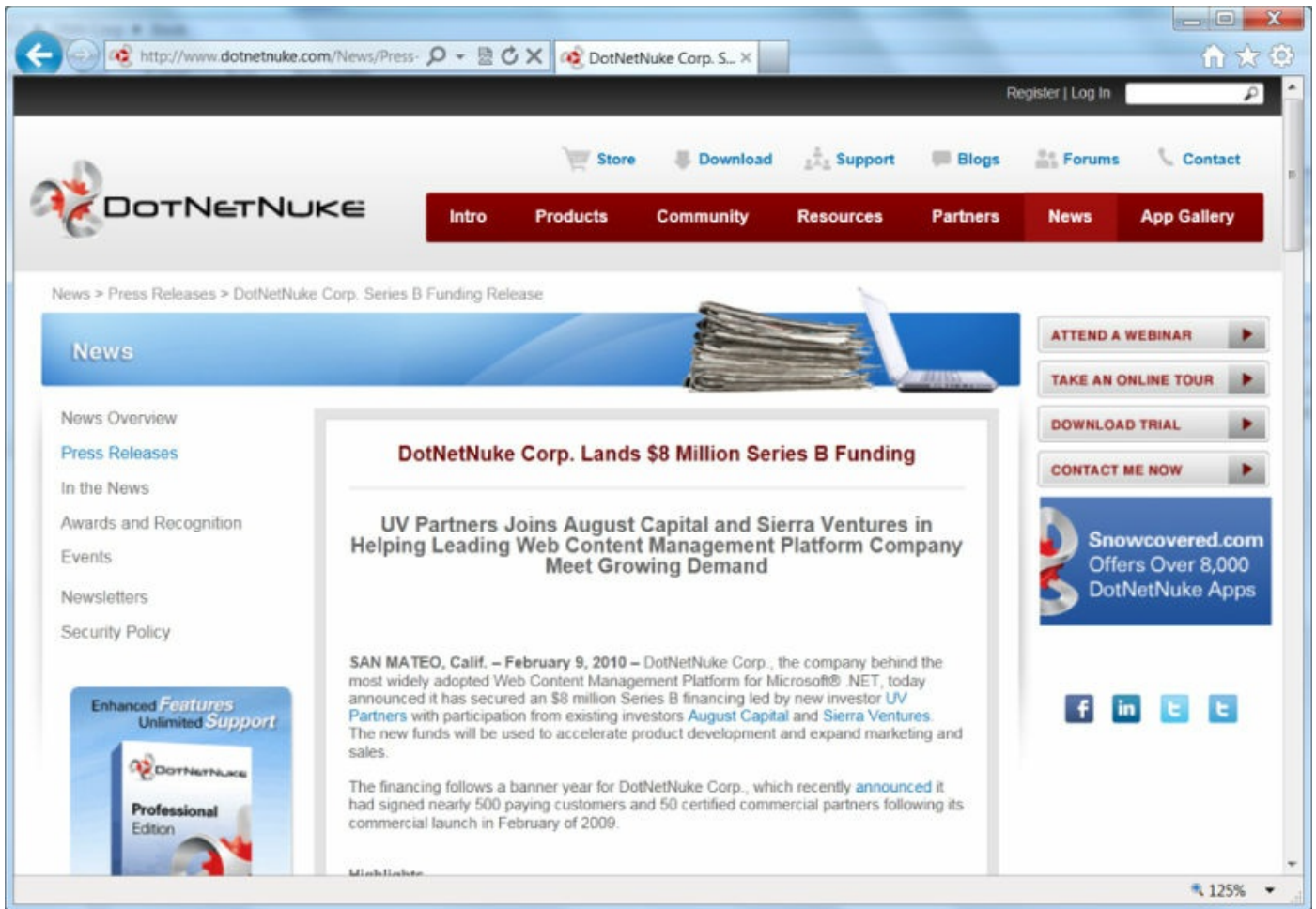


Figure 1.20

The financing round closed fairly quickly this time, and Chris Cooper from UVP joined our board of directors. In order to keep the distribution of the board in check, we also added another outside board member at this time, Frank Artale, an entrepreneur with some successful open source companies to his credit, including most recently the acquisition of XenSource by Citrix.

Open-DocumentLibrary Acquisition

The initial model of selling a commercial edition of DotNetNuke, which was essentially the same product as the open source edition except with a commercial license and professional support, was going well. However, our VP of Sales, Tom Kress, felt that we may be able to improve our conversion model if we offered some exclusive features in the commercial edition that differentiated it from the open source edition. We were operating with minimal engineering resources and managing our cash wisely, so we thought that the quickest way to add advanced functionality to the commercial edition was to acquire some complementary technology from within the DotNetNuke ecosystem.

One of the features that many customers needed was a professional document management solution. There were two dominant commercial document management options for DotNetNuke at the time: DMX and Open-DocumentLibrary. We reached out to the owners of these companies to explore the potential of an acquisition. Both vendors were interested, and after much analysis and negotiation, we ultimately decided that Open-DocumentLibrary was the best option for our requirements.

Open-DocumentLibrary was owned by Xepient Solutions, a company based in Spain, and we worked closely with it to integrate the technology into the DotNetNuke Professional Edition. This relationship with Xepient would ultimately turn into a long-term successful relationship between the organizations, as Xepient continued to be a valuable off-shore implementation partner for DNN Corp in the years to follow.

The acquisition of a commercial solution from the ecosystem was not without its share of controversy. Customers who had previously purchased Open-DocumentLibrary were concerned about future upgrades and support. And users of the Community Edition were concerned that the only way to get access to the Open-DocumentLibrary functionality was by purchasing the DotNetNuke Professional Edition—at a significant price increase over the cost of Open-DocumentLibrary on its own. Ultimately the continued availability of DMX provided another viable option for Community Edition users needing document management, and the owner of DMX benefited from us removing his main source of competition from the marketplace. However, the challenge of acquiring companies and technology from within the ecosystem would continue to be a recurring issue in the future.

DotNetNuke Enterprise Edition

By 2010, DotNetNuke was being utilized by companies of all sizes to develop websites and web applications. And although DotNetNuke Professional Edition was selling very well, it was a “one-size-fits-all” approach and did not address the specific needs of larger enterprise organizations. So in July 2010, we introduced DotNetNuke Enterprise Edition to address this market need.

DotNetNuke Enterprise Edition provided a number of benefits over the Professional Edition product offering. In addition to online trouble ticket support, it also offered telephone support for the first time, essentially the ability for a customer to pick up the phone and speak to a customer support representative in real time. In addition, we included some feature differentiation over the Professional Edition. A feature that was in high demand in enterprise environments was Content Staging, and we offered an initial version of this feature in the Enterprise Edition. The Enterprise Edition was priced at a significant margin above the Professional Edition, creating a couple of commercial options for customers to consider.

Sales of the Enterprise Edition were strong, and we made an interesting business observation in the process: Enterprise customers often choose the “enterprise” edition, not because they need the support or features, but simply because it has “enterprise” in the name. The logic is that if you are a large business, then just to be safe you should probably consider purchasing the most capable product edition because you may need some of its capabilities at some point, and it's much easier to deal with this at the initial point of procurement rather than in the future.

POET Vulnerability

In September 2010, we had to deal with a security vulnerability that was not specifically related to DotNetNuke, but rather affected all web applications that were based on Microsoft ASP.NET technology. The unfortunate situation for us is that the group who discovered the vulnerability used a DotNetNuke website as its example during an Ekoparty conference in Argentina to demonstrate how to exploit the weakness. As a result, many people were under the mistaken impression that this was a DotNetNuke vulnerability, which ended up having a negative impact on the reputation of the product.

The vulnerability was commonly referred to as “POET,” which was actually an acronym referring to a Padding Oracle Exploit Tool that was developed to break an algorithm used in the encryption of information within ASP.NET. The exploit demonstrated how someone could forge his own cookie, which would make DotNetNuke think that the user was a super user, which would obviously provide him with privileged access to all site functionality. As mentioned already, although DotNetNuke was the example application used in the exploit, this vulnerability affected all ASP.NET applications including those built on Web Forms or MVC and including Microsoft's own products such as SharePoint and Dynamics CRM.

We worked closely with the Microsoft ASP.NET team to create a patch, which blocked this attack vector and published a new version of DotNetNuke. However, we later learned that this exploit was even more fundamental to ASP.NET and that the `WebResource.axd` was vulnerable, which meant that malicious folks could even download your `web.config` file if they knew how to leverage the vulnerability. In the end, the only way to protect your website was to apply a service pack for the .NET Framework from Microsoft, and we encouraged all DotNetNuke users to do so as soon as possible.

DotNetNuke.com Overhaul

The DotNetNuke.com website had not been updated with a new visual identity for quite some time, so in 2010 we decided that we needed to modernize our online presence. Scott Willhite managed the project, and we worked closely with one of the premiere system integration companies in the DNN ecosystem, R2I, to update dotnetnuke.com.

A massive amount of work went into creating a new skin and information architecture for the site, and we managed to accomplish it without causing a major disruption in website operations or community services. At the same time that the website overhaul was going on, our director of marketing, Terry Erisman, commissioned a design firm to create a slightly modernized version of the DotNetNuke logo, and we rolled this out in conjunction with the new website (see [Figure 1.21](#)).

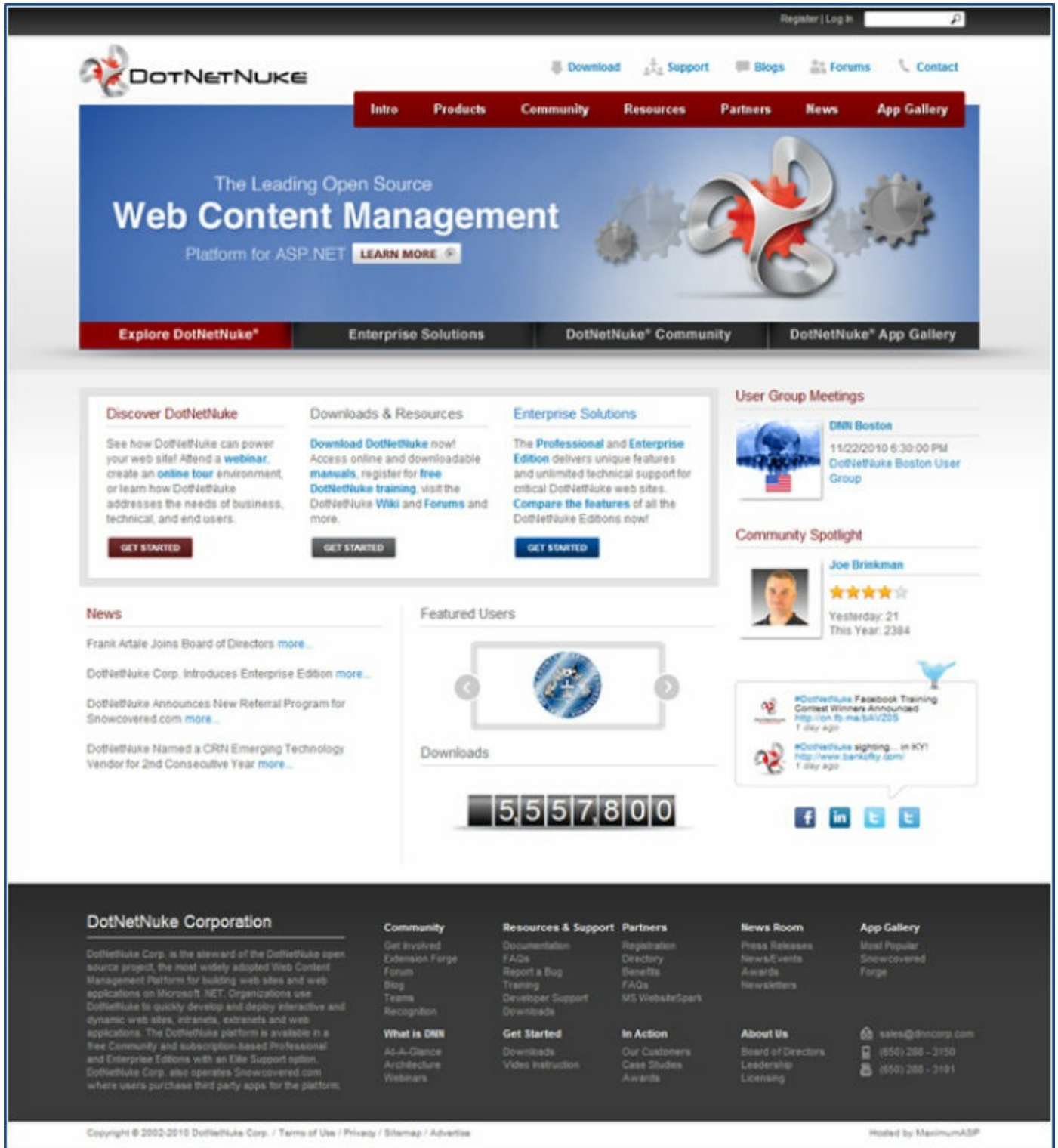


Figure 1.21

The community feedback was very positive, and this created a lot of excitement and energy in the DotNetNuke ecosystem. Our website was finally a good working example of how you could build a modern, visually appealing website using DotNetNuke technology. The website capped off a very successful year for DotNetNuke where both the business and technology felt

like they were riding the crest of a wave.

Active Modules Acquisition

Leading up to 2011 there was a lot of disruption occurring in the CMS landscape. Website builders that had previously been used only by very small businesses were becoming more powerful and starting to penetrate the mid-market. Convenience was becoming one of the most powerful forces driving procurement of software, and those CMS solutions offering capable infrastructure in addition to software were becoming a very compelling value proposition to customers. A lot of consolidation was occurring amongst the large enterprise CMS products as they struggled to maintain their dominance in an increasingly commoditized market. And the trend toward leveraging multiple devices and channels and creating an engaging customer experience was beginning to take shape. If it wanted to remain strong, DotNetNuke needed a way to differentiate itself from the rest of the CMS vendors.

Many CMS vendors had already started to introduce features related to marketing automation and analytics, so it appeared that this space was going to become crowded very quickly. Because one of DotNetNuke's strengths over its history had been related to managing online communities, focusing on the market trend around “social” appeared to be a way to stand out from the crowd and at the same time capitalize on a large industry trend.

Rather than develop a lot of social functionality from scratch, it made sense to evaluate the offerings that were already available in the DotNetNuke ecosystem, and one particular solution stood out from the rest. That solution was called Active Social, and it was developed by a company from Charleston, South Carolina, named Active Modules. Active Modules was owned by Will Morgenweck, an entrepreneur who had been creating commercial modules for DotNetNuke for many years.

After some initial conversations, we were able to determine that Will was interested in having DotNetNuke Corporation acquire Active Modules for its intellectual property. The plan was to integrate the Active Social product into the DotNetNuke CMS platform so that it could become a “Social CMS.” As part of the acquisition, Will Morgenweck also joined DotNetNuke Corporation as a product manager.

The official announcement occurred in January 2011, and once again it created a lot of controversy in the DotNetNuke ecosystem. Active Module's products were very popular, and there were a lot of loyal customers that were now uncertain about their future. We did our best to try to alleviate their

fears by being transparent with our plans, but unfortunately it took longer than expected to integrate the Active Social technology, which reduced some of the goodwill we could have achieved.

Nik Kalyani Leaves DNN Corp

One of the original cofounders of DotNetNuke Corporation, Nik Kalyani, was based in the California office and played a key role in getting the company off the ground during the start-up phase. Nik was a veteran entrepreneur and a visionary when it came to recognizing and developing high-tech business opportunities, and he had been involved in the DotNetNuke ecosystem for many years.

In the early stages after DotNetNuke Corporation received funding, there was a lot of operational effort required to scale up the company, and everyone was expected to fulfill multiple roles. Based on the fact that he was located in the business headquarters, Nik ended up focusing a lot of his efforts in the sales and marketing areas. This was somewhat unfortunate as it meant that the company lost some of the benefit of his skills being utilized in the product and technology area. And it also meant that he felt less satisfied with his role at DotNetNuke Corporation.

After a few years, Nik knew that he wanted to get more involved with creating new technology, and unfortunately the company was not yet in a position where it could satisfy his needs. In the end, he decided to move on to explore other opportunities, although he continued to remain actively engaged as an adviser for many years to come. However, as a consequence of Nik's departure, we lost a highly creative, visionary talent who had played an important role in DotNetNuke's market success.

Cloud. Mobile. Social.

After spending so much time evaluating the CMS market landscape, we had begun to establish ourselves as thought leaders in the space. The three largest trends impacting the market at the time were cloud infrastructure, mobile devices, and social engagement. Cathal Connolly, a DotNetNuke Corporation employee and one of the original Core Team members, was the first to notice that the first letter of each of these industry terms conveniently spelled C.M.S. So we were able to come up with a clever marketing slogan, “CMS Redefined: Cloud. Mobile. Social.”

This marketing slogan got us a lot of traction, and I was ultimately invited to blog about it on CMS Report and do a presentation about it at CMS Expo in Chicago in May 2011. This was the first time that our efforts were getting industry recognition outside of the software developer community. Business professionals and marketers were starting to take notice of DotNetNuke, and we were beginning to attract a new audience of users and customers.

DotNetNuke 6.0

After many successful releases of DotNetNuke 5, we knew we needed to do something bold to energize the platform and developer ecosystem. So in 2011 we set out to do exactly that.

Over the past 5 years we had noticed that the C# programming language was becoming more and more popular amongst Microsoft developers. This was reinforced by the fact that Microsoft itself provided most of its code samples and demonstrations in C#. Interestingly, a software developer from China, Ben Zhong, had utilized a language translation tool to convert the DotNetNuke application from VB to C# and had published his work on CodePlex. Due to trademark issues, we could not allow him to publish it in this manner, but rather than getting rid of it entirely, we asked him if he would be willing to maintain the C# version and continue doing the translation for each release going forward, if we agreed to publish it as an official download package from our project page. Ben agreed to this proposal, and this allowed us to gauge the interest in a C# version of DotNetNuke. The interest turned out to be very high, which ultimately resulted in us making a decision that we should adopt the C# code base as our official development branch for DotNetNuke 6.0 and for the future.

We made an announcement about the switch to C# during a Microsoft MVP conference, and it generated a lot of controversy. Some highly respected folks at Microsoft were not pleased that we created such a polarized conversation about C# versus VB, as they had been trying hard to mitigate the “language war” discussion for many years. In reality, based on the extensibility model in DotNetNuke, software developers were still free to create their extensions in either VB or C#, so were still very supportive of both communities. But switching to C# for the core framework certainly did provide some business benefits for us, as it had long been one of the blocking factors for adoption by a number of large enterprise organizations worldwide.

DotNetNuke 6 also had a full overhaul of the user experience. It adopted modal pop-ups for administrative functionality, which helped emphasize the “in-context” editing experience. And every administrative user interface was modified to use DIVs rather than TABLEs, and a whole new form pattern was introduced to provide a modern styling for the application. Cuong Dang, Ian Robinson, and Chris Paterra were instrumental in revolutionizing the DotNetNuke user experience in DotNetNuke 6.0.

DotNetNuke World 2011

In the years prior to 2011, DotNetNuke had hosted an annual user conference in Las Vegas, partnered with DevConnections. These conferences had been very successful in terms of providing exposure for the product to software developers and allowing vendors in our ecosystem to showcase their products and services. However, we felt that we may be able to achieve greater benefit and attract more attendees if we hosted our own independent user conference. We felt this was possible because we would be able to more easily cater to both a software developer and business user audience, and we would be able to charge a registration fee that was substantially less than the cost of the DevConnections event. So in November 2011 we hosted our first DotNetNuke World event.

DotNetNuke World was located in Orlando, Florida—a location chosen for its favorable weather, its abundance of conference venues, its accessibility from both North America and Europe, and its reputation as a popular tourist destination for attractions like DisneyWorld and Universal Studios. The conference took place at a popular resort hotel, and the marketing group from DotNetNuke Corporation, particularly Richard Sumas, did an outstanding job to create a larger-than-life conference experience. The conference was able to attract more attendees than in the past and generated a huge amount of excitement. However, we also gained some insight into the costs of hosting our own conference event, and the jury would be out on whether this would be something we could continue to deliver on an annual basis in the long term.

DotNetNuke 6.1 was launched during this event, and it focused on delivering advanced support for mobile devices—a capability that was becoming increasingly important for businesses catering to an online audience. The mobile device support was provided by a mobile device detection library, and DotNetNuke was one of the first CMSs to ship with this as a native feature. Initially this was based on an open source project named WURFL; however, when it adopted a commercial license with terms that were not favorable to DotNetNuke users, we switched to a library from 51Degrees.mobi.

The DotNetNuke World conference also provided a venue to showcase our many industry accomplishments and accolades we had received in 2011. We had been recognized by the Visual Studio Magazine Readers Choice Awards, Gartner Magic Quadrant for Horizontal Portals, Open Source CMS Market

Survey, DevProConnections Community Choice Awards, Packt Press Open Source Awards, and we had even reached #228 on the 2011 Inc. 500!

Providing some great entertainment, and a total surprise to myself, was the unveiling at the end of the keynote of a “bobblehead” doll in my likeness, complete with camouflage shorts, a black tank top, flip flops, and a puka shell necklace! (See [Figure 1.22.](#)) Mitch Bishop, the chief marketing officer at DotNetNuke, revealed that they had made a limited-edition run of 50 bobblehead dolls, and conference attendees were told that they could win them by doing wild and crazy antics. One person, Malik Khan from PointClick Technologies, even stripped off his clothes and dove into a pool to retrieve a bobblehead.



Figure 1.22

DotNetNuke Gets Social

After our acquisition of Active Modules in 2011, there were many questions on how we intended to utilize the technology as part of the DotNetNuke platform. The initial integration was delayed while we focused on overhauling the platform in DotNetNuke 6.0 and implementing the mobile device capabilities in DotNetNuke 6.1. So it was not until DotNetNuke 6.2, which was released in May 2012, that we were able to deliver the social functionality we had promised. This was unfortunate as we ultimately missed a window of opportunity where the social capability could have generated a more significant impact in the CMS market.

DotNetNuke 6.2 was a substantial release that included a variety of new platform features. A new Social API that was derived in a large part from Active Social offered a huge increase in functionality. Features such as social groups, friends, followers, activity stream, messaging and notifications, and an advanced user profile offered new opportunities for users and customers to build powerful community websites. These social capabilities also provided the foundation for a variety of new modules focused on user engagement.

The product launch for DotNetNuke 6.2 occurred at a DNN Partner conference event in Napa Valley, California, and was live streamed to people worldwide. Mitch Bishop, CMO of DotNetNuke Corporation at the time, had really stepped up our game from a messaging and positioning perspective and this was very obvious in the product launch presentation. Mitch had come up with the unique “Social CMS” slogan, and it resonated well with our target audience and the market in general. I presented the product slides at the launch, and Will Morgenweck demonstrated the actual product in action. We were extremely optimistic about the future of DotNetNuke and Social as an integrated product solution.

Microsoft Azure Partnership

Since 2007, we had been well aware of the industry trend toward cloud computing. It was hard to discount the convenience and benefits of leveraging a third-party infrastructure provider, and IT departments worldwide were recognizing that in most cases it did not make sense for them to procure and manage their own hardware or data centers.

In 2008, we had attempted to establish an arrangement with a large hosting provider in the DotNetNuke ecosystem that would enable us to offer a hosted DotNetNuke service. Scott Willhite was very close to formalizing this partnership; however, we had been forced to put it on hold when we secured our Series A round of funding.

In 2010, we revisited the opportunity of offering a hosted DotNetNuke solution, and Navin and Joe Brinkman visited a variety of DotNetNuke hosting providers to determine their capabilities and interest in forming a strategic partnership. Based on our long relationship with MaximumASP and its reputation as a premiere Microsoft hosting provider, we decided to utilize its infrastructure for our hosted offering. Soon after the partnership was formed, MaximumASP was acquired by CBEYOND, which resulted in some complications as CBEYOND was more focused on integrating the MaximumASP data centers and customers into its portfolio than it was in developing a white-labeled hosting service for us.

During this time, the “cloud” had become a mainstream technology trend, and a few companies were focused on providing commoditized cloud infrastructure. Amazon was offering AWS, which were essentially on-demand virtual machines that you could provision instantly and utilize as part of your business operations. Microsoft was a bit late to the game, and it was focused on delivering a specialized cloud platform named Azure that was highly scalable and capable.

Because things were not progressing as expected with CBEYOND, we decided to explore other options. Navin, Joe, and myself visited Amazon in Seattle and Microsoft in Redmond to familiarize ourselves with their offerings and road maps and determine if they were interested in partnering with us on a DotNetNuke Cloud offering.

Ultimately, Microsoft proved to be the most eager to work with us. This was driven in a large part by our participation in a Microsoft marketing program

where it was trying to improve awareness and growth of Windows-based CMS offerings. Gavin Warrener was our Microsoft contact for this program, and he played a significant role in helping us establish a strategic partnership with Microsoft based on Windows Azure. Microsoft was interested in increasing the volume of customers using Windows Azure, so it wanted us to make the Community Edition available as a cloud offering.

The partnership was officially announced in October 2012. It offered us a generous discount on Azure services, direct communication with the Windows Azure team, and assistance in promoting our cloud offering once it was available. We began working on our cloud offering and were able to hire David Rodriguez, an Azure expert based in the Canary Islands who had previously created an open source DotNetNuke Azure Accelerator product.

Making a traditional ASP.NET application like DotNetNuke function on the Azure PaaS platform was not straightforward, and we encountered many obstacles along the way. This was coupled with the fact that Microsoft was still actively developing the Azure platform, so we would run into breaking changes and compatibility issues on a regular basis. Ultimately, this is the price you pay for adopting technology early, but it did have an effect on our ability to deliver DotNetNuke in the cloud on our expected schedule.

DNN World 2012

In late October 2012, we hosted our second DNN World conference. Based on our positive experience the previous year, we again chose Orlando, Florida, as the location, but this time we held the event at a different resort hotel and conference facility.

Again, the DotNetNuke Corporation marketing team did a phenomenal job of creating the atmosphere of a huge technology event. And because the conference occurred near Halloween, a DNN Super Heroes theme was chosen, which created a lot of interesting opportunities for marketing collateral, evening events, and so on.

The conference occurred over two days with a variety of different tracks going on in parallel catering to different audiences. Scott Hunter, principal program manager for Microsoft ASP.NET, was a guest presenter for one of the keynote sessions, and Navin and I also presented keynotes.

The conference also provided a venue to announce the new DotNetNuke MVP Program. The Core Team model had served us well for many years, but it had become rather static in its membership, and there was not a well-defined process for identifying community members who deserved recognition. The DotNetNuke MVP program was modeled after the Microsoft MVP Program and was based on community contributions, primarily for activities occurring on dnnsoftware.com. The initial inductees included Stefan Cullman, Ernst Peter Tamminga, Brian Dukes, Vicenç Masanas, Clint Patterson, Ingo Herbote, Sebastian Leupold, Mitchel Sellers, Brandon Haynes, and Peter Donker.

DotNetNuke 7.0

After a number of successful DotNetNuke 6 releases, it was time for another major increment in version number. In the past, we had typically migrated DotNetNuke to the next major version number at around the same time that Microsoft introduced new versions of its operating systems and frameworks. In this case, Microsoft had released a whole wave of new technologies, including Windows 8, Windows Server 2012, IIS 8.0, Visual Studio 2012, and ASP.NET 4.5 were released in November 2012. So it certainly made sense for us to migrate to DotNetNuke 7.

Beyond ensuring compatibility for the latest Microsoft technologies, we also introduced some new capabilities in DotNetNuke 7, which ensured that it continued to remain relevant in the market. The most impactful of these changes were related to the product installation and administration experience.

A new installer was introduced, which took inspiration from WordPress's "5 minute install" and streamline the installation experience by reducing the amount of information and the number of steps required to get your DotNetNuke installation up and running. And a totally new Control Panel was introduced that was designed to look very modern and familiar, and make it more intuitive to discover the many advanced application features. The ability to personalize the Control Panel using bookmarks was added so that users could organize their most frequently utilized features in one convenient area. And the process for adding modules to a page became much simpler through the use of drag and drop.

iFinity Acquisition

Discussions had begun at the initial DNN World conference in 2011 with one of the vendors in our ecosystem about a potential technology acquisition. The product was called Url Master, and it had been developed by Bruce Chapman of iFinity based in Australia. Discussions were put on hold for an extended period of time but resumed at the second DNN World conference in 2012. URL rewriting was becoming a critical feature for web marketing professionals, so it was definitely a technology that we needed to incorporate into the platform at some point in time.

After DNN World, Bruce took the initiative to fly to San Mateo and meet with us in an attempt to move the negotiations forward. This provided a good opportunity to discuss the opportunity, and it ultimately resulted in us acquiring the iFinity intellectual property, with Bruce agreeing to join our team as a product manager.

The acquisition was announced in December 2012, and the amount of angst it created in the ecosystem was much greater than expected. This was mostly due to the fact that we announced publicly that nearly all of the advanced functionality was going to be reserved exclusively for the commercial DotNetNuke product editions. These product editions were substantially more expensive than what you were currently able to purchase Url Master for on its own. Based on the feedback, we made a concession that customers could continue to purchase Url Master as a standalone product until such time as we had fully integrated it into DotNetNuke and made it publicly available in our commercial product edition. And Bruce agreed to continue to provide support and maintenance to his existing customer base.

10-Year Anniversary

December 24, 2012, was an important date in the history of DotNetNuke. It represented the 10-year anniversary since I officially announced the open source project on the ASP.NET Forums (originally named the IBuySpy Workshop). Much had changed since those humble beginnings, and I felt exceptionally blessed for all of the friendships I had made and the impact that DotNetNuke had made on the world in the previous decade.

Chris Hammond wanted to do something special in commemoration of the big event, and I worked with him and a design firm to create an infographic (see [Figure 1.23](#)). The infographic had a theme of “metamorphosis” and showed a caterpillar evolving through the various life cycle stages and eventually emerging as a beautiful butterfly. We included a variety of significant historical events on the infographic and even embedded a few “Easter Eggs” for those people who had more intimate knowledge of the project evolution.



[Figure 1.23](#)

DNN Social

With the CMS space becoming so saturated with competition we decided that we needed to create additional specialized solutions based on DotNetNuke that could differentiate our offering. The obvious first candidate was a social solution, which leveraged the Social APIs added to the platform previously and added a variety of additional social capabilities targeted at organizations who wanted to build customer communities.

DNN Social was introduced as a commercial solution in March 2013 and included a whole suite of Social modules that could be integrated with the DotNetNuke platform. There were advanced modules that provided support for social content creation through blogging, ideation, discussions, question and answers, and events, all built on top of an analytics engine that tracked all user activities, and a gamification system that allowed you to encourage and reward community behavior. It was an impressive product release, and Chris Paterra had invested a significant amount of time and energy into bringing it to market.

However, perhaps even more challenging than creating the DNN Social product was defining the business model for it. How would it be priced? How would it be marketed? What was the customer acquisition strategy? DNN Social represented an opportunity to appeal to buyers outside of the traditional DotNetNuke ecosystem. But as most entrepreneurs know, it is a significant challenge for any company to expand from being a single product company to becoming a multiple product company. It often requires a totally different sales and marketing strategy for each product. We experienced some of these challenges almost immediately, which greatly affected the growth and success of the product in these early stages.

[DotNetNuke.com](http://dotnetnuke.com) Hacked

One of the challenges with managing a large community website is that it gets a lot of attention, and not just attention from people who are interested in contributing but also folks who have a more malicious intent.

In the spring of 2013 we had the misfortune of discovering that the dotnetnuke.com website had been compromised. It was difficult to determine the extent of the security breach, but based on our investigation, it appeared that some unknown hackers from Iran had managed to upload a sophisticated shell script to our servers. This had allowed them to elevate their user account privilege to superuser status and then utilize those privileges to gain access to other areas of our infrastructure. This was not the work of amateurs, as they covered their tracks very well. For example, they were able to bypass our IP filtering by remotely logging in to various zombie servers around the world to spoof their IP addresses. And they masked their exploits by naming their backdoor entry points as files that normally exist as part of a DotNetNuke installation and by carefully cleaning up the evidence of their handiwork as they navigated our internal infrastructure. Ultimately, it was only through careful forensic examination of our web server logs that we were able to identify and track their activities.

Our investigation revealed that it was possible that the hackers may have gained access to some of the user information on dotnetnuke.com. As a result, we decided that we needed to take immediate action to protect our community. We blocked all of the unauthorized access so that the hackers could no longer access our infrastructure. And because we had been using encrypted passwords since the website had first been launched, we decided that it was time to harden our security model, so we developed a utility that allowed us to migrate all user accounts to hashed passwords—a much more secure password protection method where there is no possible way to ever reverse engineer a string of text back into a user's password. Once this was done, we issued a bulletin to our 1 million registered users to explain the situation, urging them to change their passwords immediately.

It is important to note that the security breach was ultimately not a result of a vulnerability in the DotNetNuke application itself but rather because of an unsecure configuration in our infrastructure. We tried to be very clear about this in our public communications because we did not want to affect the reputation of DotNetNuke as a highly secure web platform.

Rebranding

In late 2012, a decision had been made that in order to redefine our identity in the market as a business solutions provider, we needed to do some substantial rebranding. The common buzzword in Silicon Valley to describe a major shift in business approach is “pivot,” and in the first half of 2013 we worked with a number of third-party consultants from the Bay Area who helped us with our overall brand strategy and execution.

The rebranding project was a massive undertaking. It included everything from our company name and logo, to our website and email domain names, to our product names, marketing collateral, website content, and visual appearance. A lot of time and energy went into this activity, but in order for it to not be a disruptive distraction while the work was in progress, very little information was shared either internally or externally while it was going on.

From a high level, the most fundamental change was moving away from the “DotNetNuke” brand and fully embracing “DNN.” The rationale was that the “nuke” reference has never been particularly positive or professional, and it made sense to distance ourselves from legacy systems that shared the common branding bond, such as PHP-Nuke. In addition, there was uncertainty about how long Microsoft would continue to utilize the “.NET” branding as part of its own technology platform (after more than a decade, there were some rumors circulating that a new brand strategy might emerge). We already owned the trademarks for DNN, and it was already getting widespread usage throughout the ecosystem, so it made sense for us to concentrate our efforts on this brand. As a result we needed to migrate all references of DotNetNuke to DNN, including critical infrastructure items such as our website and email domain names. We were able to acquire the domain dnnsoftware.com from a member of our ecosystem, and this would become our new online identity going forward. We also officially changed the name of the company from DotNetNuke Corporation to DNN Corp.

When the decision was made to utilize DNN for our branding, it also prompted a decision to create a new logo. The current logo had been in use since Nik Kalyani created it back in 2005, and it was time for a significant overhaul. Parker Moore, a creative firm from the Bay Area that had previously worked with Apple among many others, came up with a variety of logo concepts before we chose a simple “D” design. The new logo was simple, yet modern, and very adaptable to a variety of visual treatments. It also included

a new color palette that would need to be utilized in all of our future marketing collateral. See [Figure 1.24](#).



[Figure 1.24](#)

From a product perspective, the move away from DotNetNuke meant that all product editions also needed to be rebranded. The feedback from the consultants was that the commercial editions needed to have a brand that was differentiated from the open source platform. So we embarked on an exhaustive journey to come up with a new commercial product brand. The goal was to come up with a brand that emphasized the “genuinely empowering” theme that had come out of earlier brand identity strategy discussions. Many names were suggested, but after doing research we would find trademark issues or other conflicts. In the end we chose the name Evoq, a unique spelling variation of the word evoke that means “to call up or produce (memories, feelings, and so on).” So the commercial products would be branded as Evoq X, where “X” would refer to a specific business solution, while the open source product edition would be branded DNN Platform.

At the same time that the branding and logo activities were going on, a major overhaul of our website was also underway. The goal of this overhaul was to better promote the commercial solutions while still maintaining a strong community presence. A totally new information architecture was developed, and once the new logo and color palette were approved, work also began on a modern new skin. The approach with this website overhaul (which was different than what we had ever done in the past) was that rather than upgrading the existing site, we were going to create a completely new installation and only migrate the content that made sense as part of the new information architecture. The logic was that over the course of a decade, the DNN website had acquired a lot of “baggage,” and a lot of benefit could be obtained by starting with a fresh new website foundation.

The website was launched in July 2013 (see [Figure 1.25](#)), which also served as the promotional vehicle for announcing the rest of the rebranding changes. In general, the community response was very positive, especially from those

folks who relied on DNN for their livelihoods (as they had long been the people telling us that the “nuke” name often caused them difficulties in business engagements). However, there was also fear and uncertainty expressed by a number of longtime community members who did not feel comfortable with the DNN Platform branding and who felt the rebranding represented a shift away from treating the open source project as a first class citizen. Ultimately, this would result in the formation of DNN-Connect, a European nonprofit group whose public mandate is to ensure the longevity of the DNN open source platform.

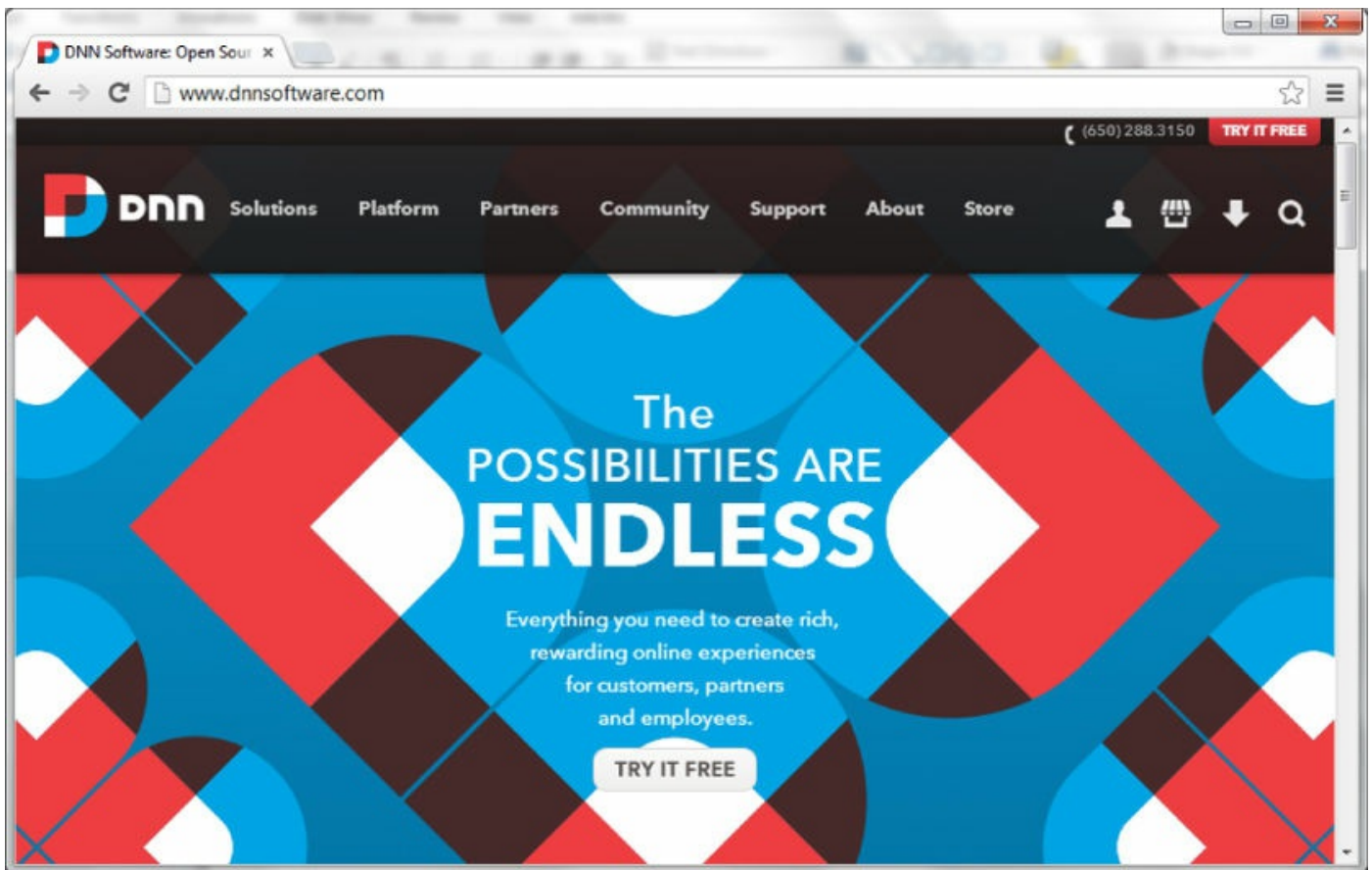


Figure 1.25

One of the unfortunate side effects of the rebranding and approach taken with the website overhaul was that we lost significant SEO and web traffic. Specifically, the change in website domain name and the new information architecture resulted in a lot of broken links for visitors coming to the site from search engine referrals. In addition, a lot of historical content about DNN from the past decade was not included in the new website and therefore started to be flushed from the search engine indexes. To some extent these side effects were expected; however, the magnitude of the impact on traffic

was much larger than expected.

DNNCon

After reviewing the financial aspects of the DNN World conference events, DNN Corp determined that it was not viable to host another DNN World conference event in 2013. We announced this news publicly and hoped that the void could be filled by some community events.

Arrow Consulting and Design, a DNN Partner and solutions provider based in West Palm Beach, Florida, graciously stepped up and volunteered to host a large scale DNN event in November 2013 in its home city. The principal owners at Arrow, Ryan Morgan and Raul Rodilla, had long been proud supporters of DNN and based on the recent branding changes in the ecosystem, they spoke to Will Strohl and the Day of DotNetNuke events were rebranded to DNNCon.

DNNCon was a very successful user conference. It had a full day of training followed by a full day of conference sessions. It also had the ever popular DNN After Dark, an evening social event where conference attendees could get together, socialize, and get to know one another better. The sponsorships were so generous for the event that Arrow was able to donate more than \$10,000 to a local charity called Place of Hope at the conclusion of the conference.

DNNCon (and its sister event in Europe, DNN-Connect) provided the perfect venue to publicize and reward each batch of DNN MVPs. In 2013 and 2014, the following MVPs received recognition for their exemplary community contributions: David Lee, Gilles Le Pigocher, Jason Brunken, Michael Tobisch, Roger Selwyn, Scott McCulloch, William Severance, Chris Hammond, Gifford Watkins, Robb Bryn, Allen Foster, Peter Donker, Ernst Peter Tamminga, Brian Dukes, Vicenç Masanas, Mitchel Sellers, Sebastian Leupold, Matthias Schlomann, Timo Breumelhof, Torsten Weggen, Oliver Hine, Wes Tatters, Sacha Trauwaen, Will Strohl, Jay Mathis, Geoff Barlow, Julien Girerd, Bogdan Litescu, Daniel Mettler, Erik van Ballegoij, Richard Howells, Scott Wilkinson, and myself.

Scott Willhite Moves On

In December 2013, another cofounder decided to move on from the company. Scott Willhite was one of the original Core Team members and had worked closely with me ever since 2003, providing valuable wisdom and leadership to the project as well as shouldering a significant portion of the workload. Scott was based in Seattle, Washington, and due to personal commitments, he continued to work remotely even after the company received funding. This was challenging for him, but he still made many significant contributions to the company.

Over the course of a decade, Scott played a key role in the success of the DNN project. He was not a person who ever looked for individual recognition or gratitude but preferred to work behind the scenes and take responsibility for any area that was being neglected or underserved. This character trait meant that he often ended up volunteering for time-consuming, operational tasks that were essential to the company and project but provided very little personal satisfaction or reward. He also felt very strongly about community and the role that DNN Corp played as stewards within the open source ecosystem.

Scott's official title was director of community relations, and as the company grew and adapted to the changing market landscape, the focus on community also began to change. The new model for managing community no longer aligned as closely with Scott's personal perspective, motivations, or strengths. As a result, he felt that it was time to move on and he tendered his resignation. And although I knew it was the right decision for Scott, I still had a very difficult time dealing with it because of our long-term friendship and trusted working relationship.

DNN 7.x Releases

Throughout 2013 and 2014, DNN Corp continued to deliver new major versions of DNN on roughly a six-month release cadence. These releases followed the DNN 7 naming convention and included many new features for both the commercial editions as well as the open source platform. Integration of the intellectual property acquired through the iFinity Url Master acquisition occurred in DNN 7.1, which greatly improved the support for SEO friendly URLs. Advanced digital asset management was added to the platform and security improvements in regard to user passwords were introduced to improve the default product configuration. A new architecture for indexing site content was added to DNN based on the popular Lucene open source project. The user experience received some important updates in administrative scenarios so that it could better accommodate the large volumes of information that our users and customers were managing. And a large focus was placed on improving the overall performance of the application, resulting in significant gains in page load times and reduction of HTML payload sent to the browser.

During the development of DNN 7.x, major changes also occurred in the product infrastructure. From a source code management perspective, we had been using Team Foundation Server for a number of years, and although it met our needs for internal development, it did not support an open source development model. Specifically, it did not provide a mechanism for allowing people to browse the source code in real time or a way for people to make source code contributions that could be easily integrated with the platform. Recently, a new source code management system named Git had become the de facto standard for managing open source projects, and we decided that it made sense to migrate our code base to Git and host our open source project repository on GitHub. From an issue management perspective, we had been using Countersoft's Gemini product for many years, but it had a variety of limitations, and upon review it became obvious that there were other software products that were much more popular and better suited for open source style development. We chose to migrate to Atlassian's JIRA product and utilize its hosted offering.

My Departure from DNN Corp

In August 2014 I made an announcement that shocked many folks within the DNN ecosystem. In some ways, it was even a shock to myself. DNN Corp and I had decided to part ways. As is common in these types of situations, the public details surrounding my departure were purposely kept to a minimum. The final farewell blog post I wrote was reviewed and edited extensively by the company prior to its publication to ensure that it was professional and consistent with the messaging that was agreed upon by both the company and myself. As a result, it was one of the shortest and most impersonal blog posts I had published over the lifetime of the project. So, not surprisingly, it fueled further discussion and speculation by the community in regard to the real story behind my withdrawal from the company.

The reality is that I had been feeling frustrated for quite some time. I was not frustrated by the progress on the open source project but rather by the challenges of trying to preserve the delicate balance between the open source ecosystem and the commercial needs of the company. As project founder and steward, I shouldered the majority of the responsibility over the years in terms of trying to ensure harmony between the various stakeholders and their competing interests. This responsibility put me in the middle of every debate and every conflict, which ultimately weighed on my conscience and was extremely draining from an emotional perspective. Basically, the main challenge in trying to ensure a balanced approach is that there is never truly a win/win outcome for all parties. In every instance there is at least one party who feels that its perspective or needs were not fully realized. So in trying to please everyone, you usually feel like you are pleasing no one, and you start to feel increasingly isolated and alone. The key, of course, is to not focus on each instance but rather to look at things from a higher level and try to ensure that the net result of all of the decisions combined are balanced. Over time, if stakeholders feel like they lost some battles but won others, it will preserve their faith and trust in the environment. However, it is next to impossible to satisfy everyone. The only thing you can do is develop a “thick skin” and objectively try to constantly live up to the fundamental community ideals. This is all easier said than done, and over time I felt like the philosophical divide between the open source community and commercial interests of the company were becoming wider and that I had begun to lose my influence on the tug-of-war between the various stakeholders in the DNN ecosystem.

Another challenge faced by many organizations is related to creating a winning culture. The ultimate goal is to create an environment where everyone understands the guiding principles and vision and are all working together in unison toward achieving that vision. In the early stages of DNN there was a strong founding leadership group who had been together from the very beginning of the open source project. And based on our publicized community ideals as well as our track record of how we had conducted ourselves over time, we had created an almost cult-like atmosphere within the DNN ecosystem. Very few technology companies can rival the number of loyal followers that DNN had attracted in the first five to six years of the open source project—the passion and enthusiasm of our third-party evangelists was unparalleled. One key to creating this environment was the “abundance mentality”—the belief that the project and organization would be successful only if the other members of the ecosystem were also successful. This is an extremely powerful concept, as once the trust and credibility had been established, it created an army of loyal followers who helped facilitate the viral growth that followed. Word spread that the DNN ecosystem had no barrier to entry and was ripe with opportunities for everyone, regardless of their background.

When DNN Corp received venture capital funding in late 2008, we were lucky enough to be able to hire many key evangelists from the community. This further strengthened the core of the company, both technically and culturally. People were excited to work for DNN Corp because they wanted to be able to contribute in a meaningful way to the ecosystem that they already knew and loved. DNN Corp benefited from this atmosphere and achieved a couple years of phenomenal commercial growth and success. However, it is very difficult for any organization to sustain this type of momentum in the long term. The biggest challenge is related to employee turnover. As founders and employees that had been recruited from within the DNN community moved on to other opportunities, the focus of the organization began to change. It became more commercially oriented, and new employees entered into a new culture with different priorities. In addition, the leadership of the company changed a number of times, and each transition brought new personalities who wanted to make an impact on the organization—sometimes in ways that realigned the organization's goals. This created challenges and shifted the company's identity.

This was difficult for me, as I felt that my role and perspective during this

time were often misunderstood. My heart and passion had never wavered from the open source DNN project; however, I also wanted the company to be successful. I had opinions on where I felt the opportunities existed for DNN to move forward so that it could continue to serve both the community and commercial goals. And although my thoughts were not always consistent with other folks on the leadership team, I did my best to support the ultimate direction of the organization. I felt that the most important contribution I could make was to try to ensure that the organization maintained the most positive aspects of its historical identity so that the community would feel comfortable in the transition and the company could retain its hard-earned goodwill and reputation. I was still seen as the public face of the company and the spokesperson of the DNN community, so I knew I would assume the primary responsibility of communicating our intentions and goals clearly and transparently to the ecosystem. This was more challenging than it sounds, mainly due to the fact that my own personal identity and reputation had become so intertwined with the open source project. In the end, we did succeed in establishing a new project identity, and I am proud of my contributions toward that achievement. However, in the process I had succeeded in straining a number of critical relationships that resulted in me feeling less connected with the team. This ultimately led to the announcement on August 12, 2014 (see [Figure 1.26](#)), that the company and I had agreed to part ways. I gave up my role as CTO and also resigned my seat as a member of the DNN board of directors.

Turning The Page To Begin A New Chapter In My Life...

Aug
12

[f](#) [t](#) [e](#) [+](#) Comments (29), [Permalink](#)

There is a famous quote that "if you love something, set it free", so it is with immense pride and emotion that I acknowledge that it is time for me to step aside from the open source project, company, and legacy that I helped create and transition my responsibilities at DNN Corp.

After nearly 14 years of working on one of the most widely deployed open source CMS platforms, the company and I have concluded that it's time for me to seek new opportunities that build on these experiences. My passion for building products and ecosystems that make an impact on the world is as strong as ever and I look forward focusing my energy and creativity along similar lines.

During my tenure at DNN, the open source project and company achieved great success and received many industry accolades - some of which I would have never dreamed remotely possible at its humble inception. The company has grown into a mature enterprise with high quality products that are mission critical to customers worldwide. I have also been blessed with establishing many genuine relationships and trusted friendships within the DNN ecosystem which I know will endure forever.

Figure 1.26

There is one last point of distinction in regard to my departure that I think is important to understand. Similar to any standard working relationship, my association with DNN Corp was based on an employment agreement that had a specific start date and, upon my departure, an end date. As a founder, I did have equity in the company, and I continue to own those shares as they were fully vested during my tenure with the organization. On the other hand, the DNN open source project existed prior to the formation of DNN Corp, and based on its open source license, it will continue to exist indefinitely. So although I left DNN Corp, I have not left DNN. My heart and passion remain committed to this vibrant open source community that I started over a decade ago and that will always remain a proud part of my legacy. As a community member I am free to contribute to the open source project and ecosystem in the same capacity as every other community member, and I intend to do so as much as my new life and responsibilities allow. I continue to be thankful and grateful every day for the positive impact I was able to make on this world and for all of the sincere, lasting personal friendships I have earned as a result of this open source community. At the end of the day, these are the

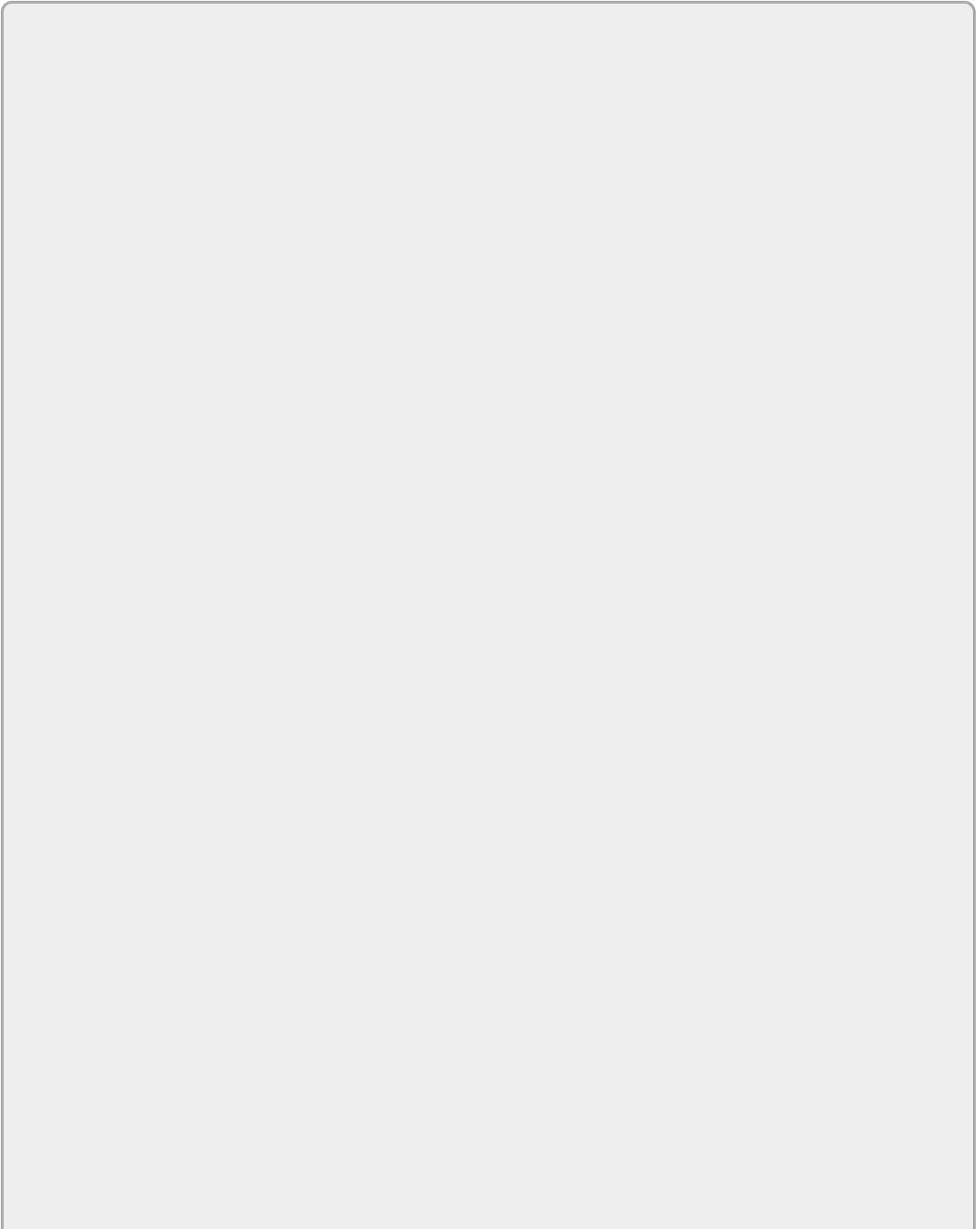
most valuable riches I could ever hope to receive.

Summary

DNN is an evolving open source .NET CMS platform. The organic community ecosystem around DNN is vibrant and dynamic, providing the essential environment for long-term growth and prosperity. You will always be able to get the latest high-quality product release, including full source code, from <http://www.dnnsoftware.com>.

Chapter 2

Installing DNN Version 7



What You Will Learn In This Chapter

- Preparing your system to install DNN version 7
- Installing DNN version 7
- Upgrading from previous versions of DNN
- Getting a trial version of Evoq Content

This chapter reviews the installation process of the DNN Platform and shows how to get a trial version of Evoq Content. If you want to create a website, you have different options to get started.

- The Install Package contains only the files needed to run a website in IIS (Internet Information Services). These files are compatible with SQL Server and SQL Express.
- The Source Package contains the source code of the application including every module included in the DNN Platform.
- The Deploy Package is used by the Web Platform Installer and includes the necessary files you need to deploy to a web server. The Web Platform Installer also verifies and installs additional dependencies to get the website up and running with little effort and knowledge.
- The Upgrade Package contains only the files needed for an upgrade of an existing website. Your previous edition for the upgrade must be version 6 or higher. Notes on upgrading from earlier versions are available at http://www.dnnsoftware.com/wiki/page/suggested_upgrade_path.
- Evoq Content Trials are hosted in Microsoft's Azure servers and are the best way for you to test the advanced functionality without having your own infrastructure or involving your IT department.

There are four options to install the DNN Platform, and you're probably wondering, which one is for me? If you're a developer interested in extending the functionality of the application, you may be interested in the Source Package; however, you should start investigating creating DNN extensions (modules, providers, etc.) before modifying DNN's core code. If you have a server ready to host your website, the Install Package is your best option; if you have a new server and you're not sure if you have all the dependencies to

successfully run a website, you can try the Deploy Package; and finally, if you have an existing website and you want to upgrade to the latest version of the DNN Platform, you should use the Upgrade Package.

When do you use an Evoq Content Trial? Evoq Content includes advanced functionality like workflow, digital assets management, and advanced permissions among other features. If you want to run a professional website, you may want to try Evoq Content in the cloud, which takes seconds to get set up.

What You Need To Install DNN Platform Version 7

The two basic ways to install DNN Platform 7 are using the installation package or using the Web Platform Installer. Before you begin, make sure your system requirements are sufficient for the installation.

- Web Server: IIS 7.0 or higher (contained in Windows Vista and later for desktops, and Windows Server 2008 and later for servers)
- Microsoft .NET Runtime: ASP.NET 4.0 or higher
- Database: Microsoft SQL Server 2008 or greater

Installing the DNN Platform Using the Installation Package

If you want to install the DNN Platform 7 Install Package, follow these steps:

1. Download the Install Package.
2. Unzip the package.
3. Create a database in SQL Server.
4. Create a database account.
5. Configure IIS.
6. Set file and folder permissions.
7. Perform the installation.

Step 1: Download the Software

Navigate to <http://www.dnnsoftware.com/Community/Download>, which is under the Community section of DNN's website, as shown in [Figure 2.1](#). Click the Download button under the Install Package section. This redirects you to <https://dotnetnuke.codeplex.com/>, and the install package starts downloading automatically.

Download DNN Platform

[DNN Forge](#) [Manual](#)

Build amazing websites. Download the Free, Open Source DNN Platform!

Choose the DNN installer that's right for you:

STANDARD PROCESS

DOWNLOAD DNN PLATFORM
Install and configure a DNN website on your desktop now!

MS WEB MATRIX

DOWNLOAD DNN PLATFORM
Install and configure a DNN website in **WebMatrix** with Microsoft's free, lightweight web development tool, a great way to learn DNN extension development basics.



Or just download the open source files you want.

INSTALL PACKAGE

Just the files needed for a runtime deployment to a web server.

Download the Install Package if you want to deploy a live site to a web server with the minimum required files.

[Download](#)

SYMBOLS PACKAGE

PDF and XML files that aid in DNN module development and in DNN debugging.

Download if you are a developer in need of platform symbols.

[Download](#)

SOURCE CODE

Full application source code.

Download if you are a web developer interested in the DNN Platform web application framework source code.

[Download](#)

UPGRADE PACKAGE

Files needed for upgrading an existing installation of DNN Platform, (no module packages)

Download this package to upgrade an existing installation to a newer version.

[Download](#)

API HELP FILE

A compiled help file with DNN API documentation.

Download if you are a developer who wants to understand the DNN Platform API.

[Download](#)

And then install some extensions!



Some of DNN's most popular extensions are freely available from open source developers in the **DNN Forge**.



Commercial extensions from application vendors can be purchased in the **DNN Store**. There are thousands of products available from sites to enterprise grade applications.

The DNN Promise.

DNN OPEN SOURCE LICENSE TERMS

DNNP - <http://www.dnnsoftware.com>
Copyright (c) 2000-2013
by DNN Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

ABOUT DNN

DNN Community (DnnSoftware) provides a suite of software for creating web, marketing online, applications for enterprise, learning and professional DNN technology is the foundation for DNN's web site solutions and our partners.

Includes: User, Value, Hardware, Data, Content, Membership, User, Web, Services, (DNN), DNN, NACAR, Business Health and the City of Dnn.com. In addition to our commercial solutions, DNN is the steward of the DnnSoftware Open Source Project.

FOCUS

Service
User-Centric
Data-Driven

DNN PLATFORM

DNN Architecture
DNN Community
Download DNN
Community Blog

DNN NEWS

Case Study
Press Releases
Articles

RESOURCES

Webinars
DNN Manuals
DNN Whitepapers
DNN Tutorials

CONTACT

Support and Sales
Work at DNN
Subscribe

[Figure 2.1](#)

Step 2: Unzip the Package

Extract the entire contents of the Zip file to your chosen installation directory. To install on your local system, you can place your website under the C:\inetpub\wwwroot folder, like in this folder:

C:\inetpub\wwwroot\DNN7. If you have a hosting account and you want to install the DNN Platform on your new server, make sure you read the instructions of your hosting provider and follow the steps to upload the package.

Step 3: Create a Database in SQL Server

If you're using a hosting service, your hosting service very likely comes with a preconfigured SQL Server database, and your provider will give you detailed instructions on how to connect and use it.

Otherwise, open SQL Management Studio, right-click Database, and select New Database, as shown in [Figure 2.2](#).

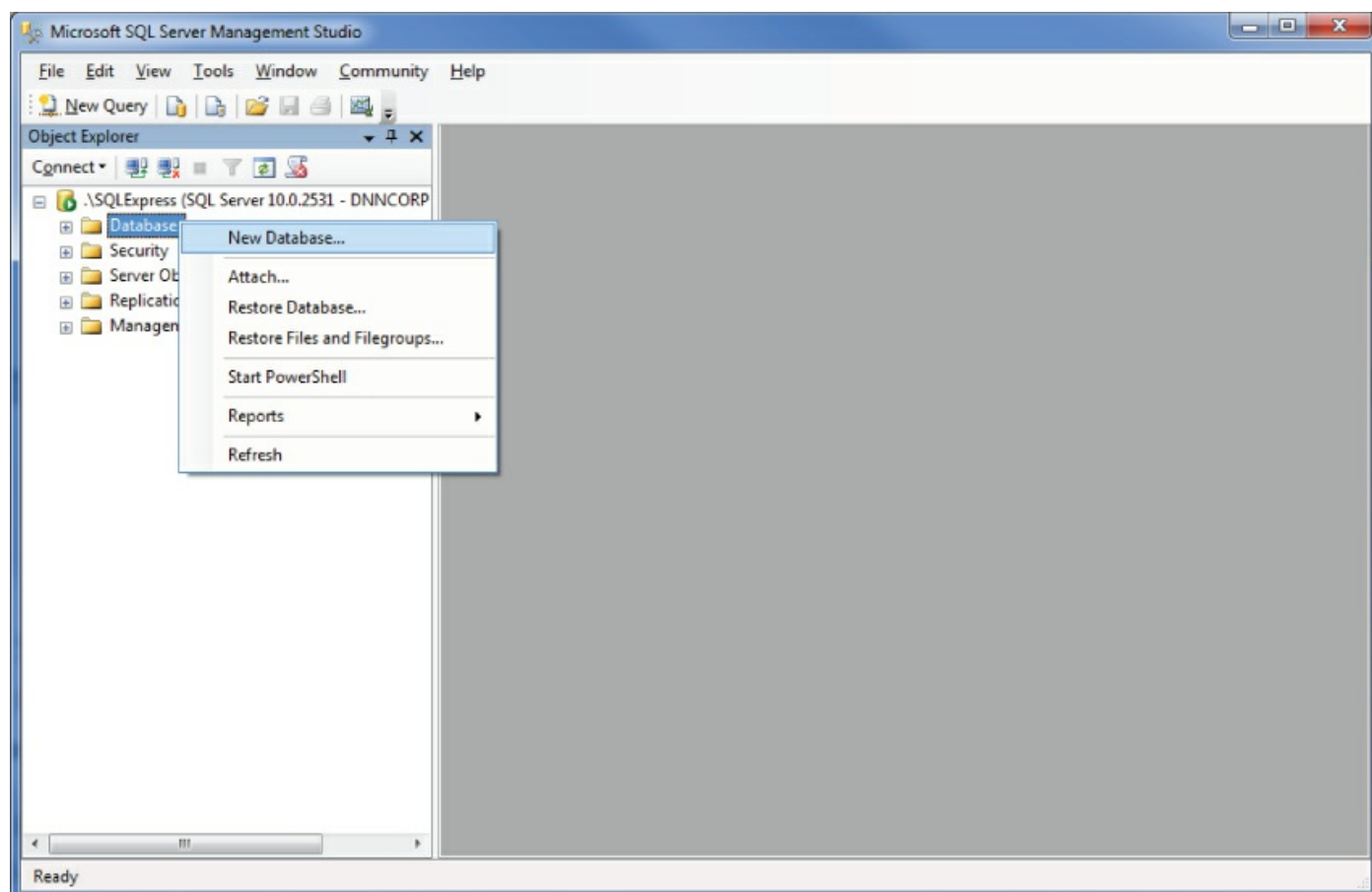


Figure 2.2

Give the database a name and click OK.

Step 4: Create a Database Account

There are two main options for creating an account in SQL.

- **Windows Security:** This method uses the account under which your website is running. This option is very secure, but it's often not supported in hosting environments.
- **SQL Server Security:** Access the database using a username and a password.

Step 5: Configure IIS

The next step is to create a website pointing at the DNN Platform installation files. Follow these instructions to complete the configuration.

Click the Start icon, type **inetmgr** in the search box, select Internet Information Services (IIS) Manager, and open the program. You can create a new site, which would have its own URL, or you can create an application under Default Web Site, which will have a URL under <http://localhost> (e.g., <http://localhost/dnn>). Many in the DNN community use the URL <http://dnndev.me> (or a subdomain, like <http://mysite.dnndev.me>), which is set up to point to your local computer. In that case, you can right-click the Sites folder and click Add Website. Here you can enter the site name (e.g., dnndev.me), the physical path to the website (e.g., C:\inetpub\wwwroot\DNN7), and the host name (e.g., dnndev.me). Notice that a new application pool based on your entered site name is automatically created, though you can choose an existing one if you want.

Alternatively, if you want to create an application under Default Web Site, you can click to expand Sites, right-click Default Web Site, and click Add Application. Type the alias of the site (e.g., **dnn** for <http://localhost/dnn>), select an application pool (ASP.NET v4.0 or .NET v4.5 would be good choices), and select the physical path to the DNN Platform location.

Step 6: Set File and Folder Permissions

To use IIS you need to set file permissions on the folder where you plan to install the DNN Platform. To set file permissions, you need to know the

identity used by the process to run your site so that you can give the permissions to the correct identity. In IIS, select Application Pools to view a listing of the available application pools. Find the pool assigned to the website or application you created in step 5. If its Identity column says ApplicationPoolIdentity, this means a new identity was created just for that application pool (e.g., the application pool dnndev.me has an identity of IIS AppPool\dnndev.me). Otherwise, it lists the name of the identity used by that application pool's worker process (probably NetworkService).

Once you know the identity to assign permissions to, you can right-click the website's folder (e.g., C:\inetpub\wwwroot\DNN7) and click Properties. Now click the Security tab. See [Figure 2.3](#).

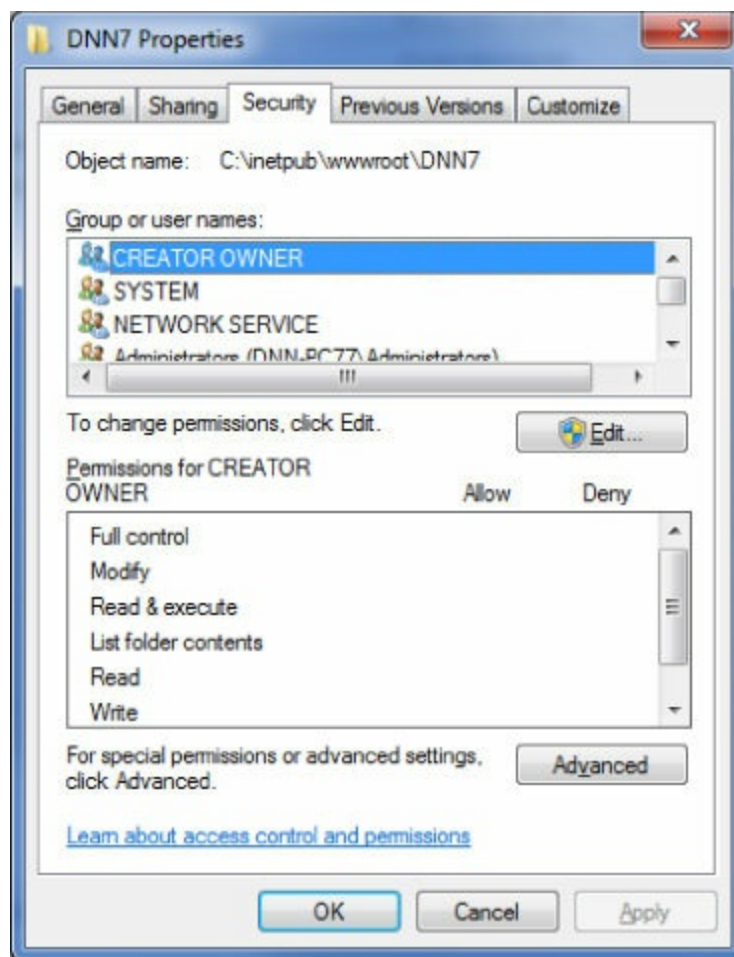


Figure 2.3

Click Edit ➔ Add. If the application pool is using ApplicationPoolIdentity, make sure the local computer is selected as the location; then enter the full identity name (IIS AppPool\ and then the application pool name). If a different identity is used, you can enter it (add a space in “Network Service”)

or click Advanced and click Find Now to pick from a list (the IIS AppPool identities are “virtual accounts” and don't show in that list). Highlight the identity you just added, check the box at the intersection of Allow and Modify, and then click OK three times. This configures IIS to be able to modify files in the website folder when DNN wants to do so.

Step 7: Perform the Installation

In the Internet Information Services Manager, right-click the newly created website or application, scroll down to Manage Web Site, and select Browse from the secondary menu. See [Figure 2.4](#).

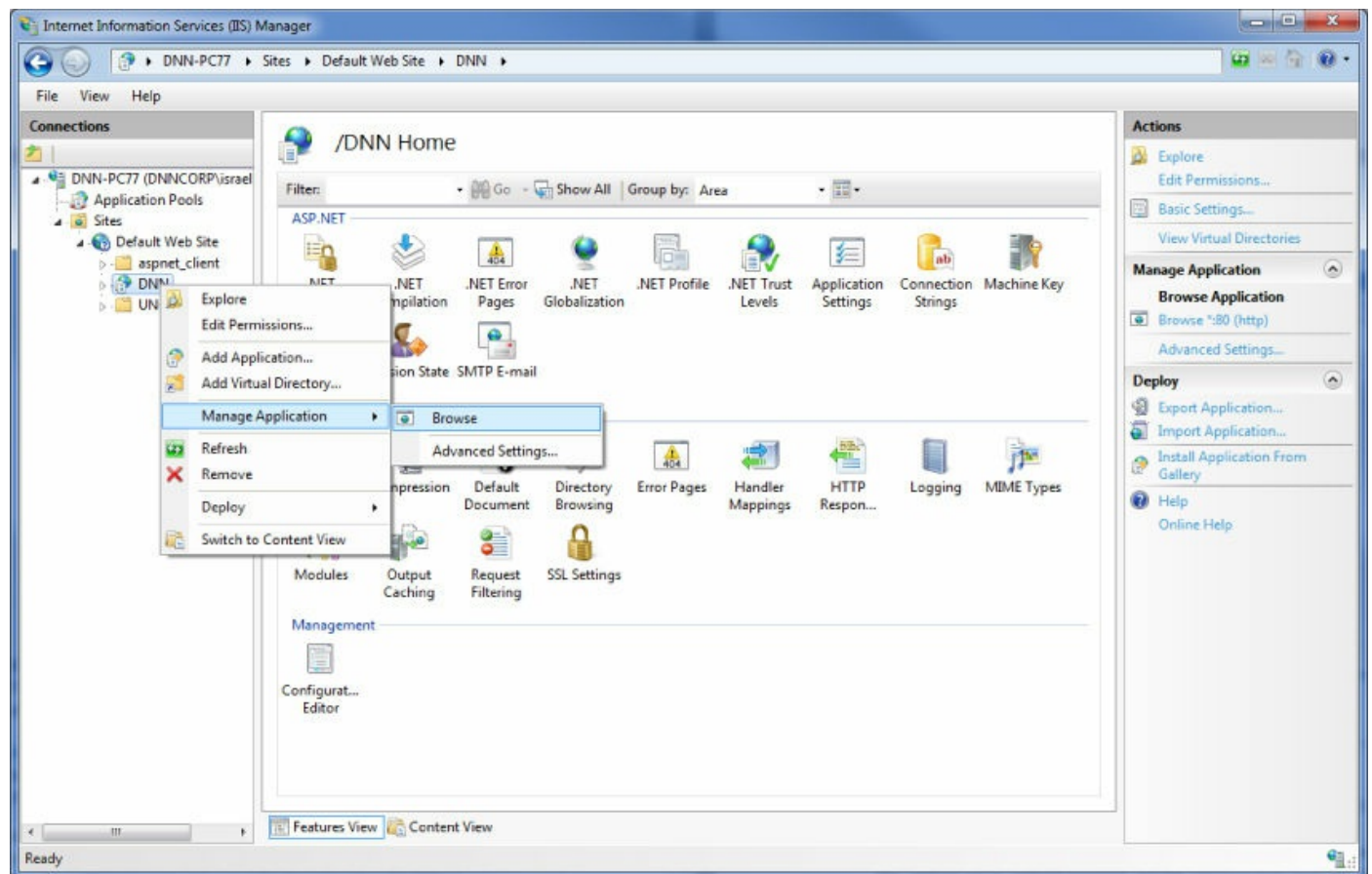


Figure 2.4

When the browser opens, you'll see the installation screen. See [Figure 2.5](#). In this screen you have the options described next.

- **Administrative Information:** Here you need to provide the username and password for the initial host user (i.e., a user with unlimited permissions to the entire DNN installation).
- **Website Information:** In this section, you can provide some custom

options for your website, such as website name, website template, and administration language. The administration language is the one used when working on the website.

- **Database Information:** If you decide to use the default option, your website will be using an SQL Express database included in the installation package. But you can also create your own SQL Server or SQL Express databases and provide the information needed for the system to connect to it.

Installation

1 Enter Your Account Information

2 Proceed with Installation

View Website

To setup your Installation, enter the following information. [View Installation Video](#)

Administrative Information

Username * ⓘ

Password * ⓘ

Confirm * ⓘ

Website Information

Website Name * ⓘ

Template ⓘ

Language ⓘ

Database Information

Database Setup ⓘ Default Custom

Database Type ⓘ SQL Server Express File SQL Server/SQL Server Express Database

Server Name * ⓘ

Filename * ⓘ

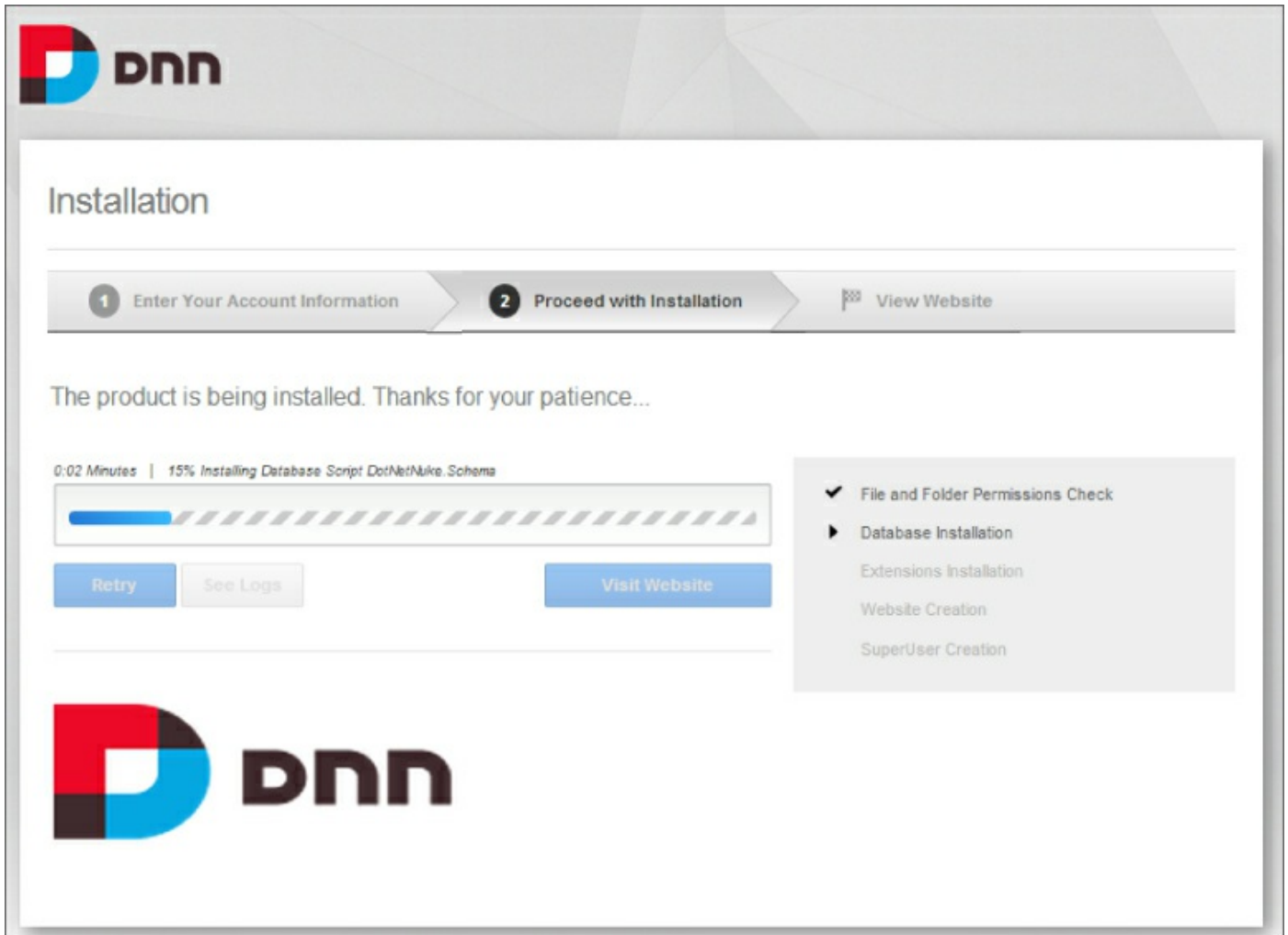
Object Qualifier ⓘ

Run Database As ⓘ Database Owner

Continue

Figure 2.5

After entering the information, click Continue, and the system shows you the Installation In Progress screen. See [Figure 2.6](#). The installation usually takes less than a minute, and then the Visit Website button becomes active and you have a new website. See [Figure 2.7](#).



[Figure 2.6](#)

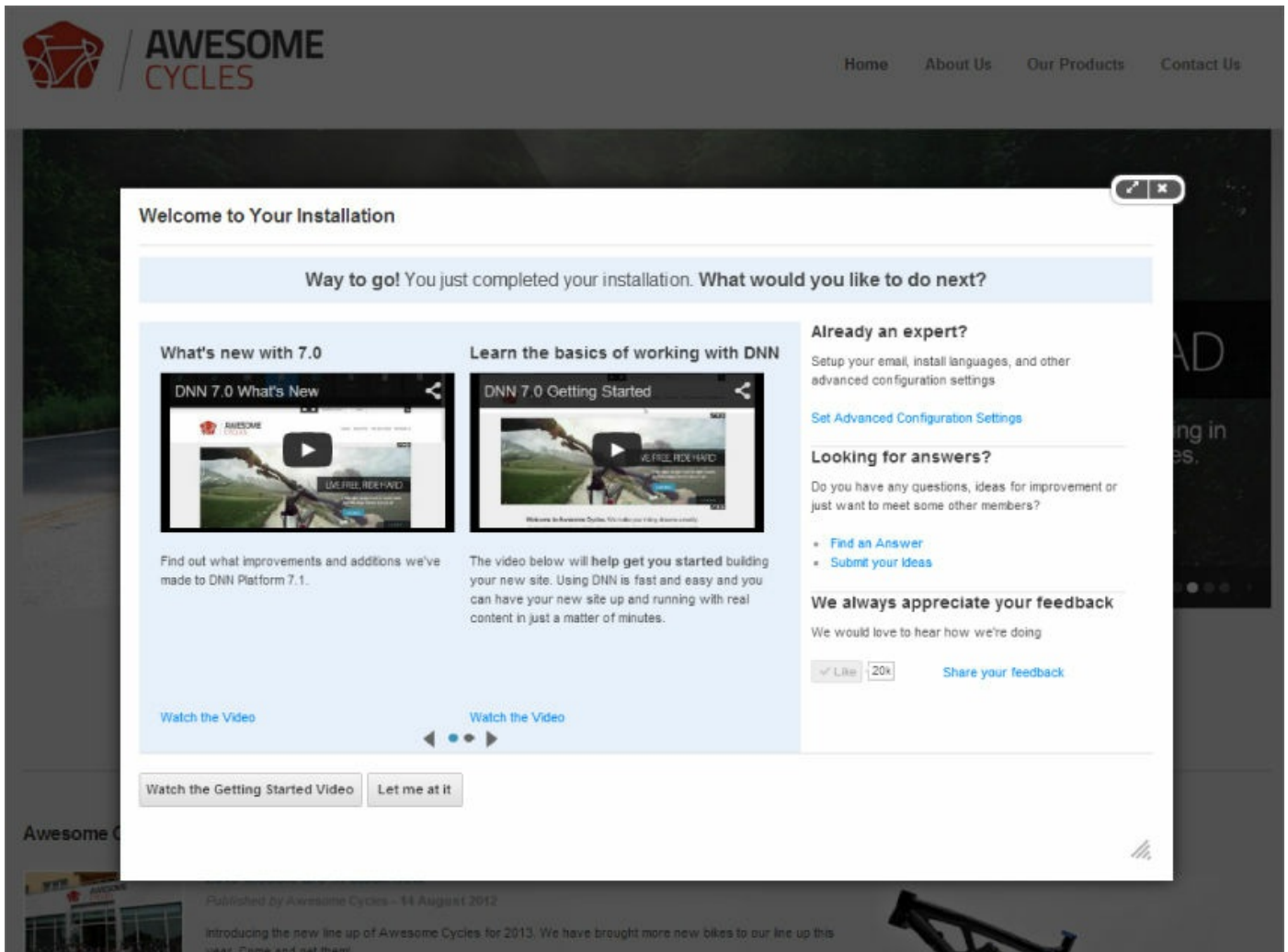


Figure 2.7

Installing the DNN Platform Using the Web Platform Installer

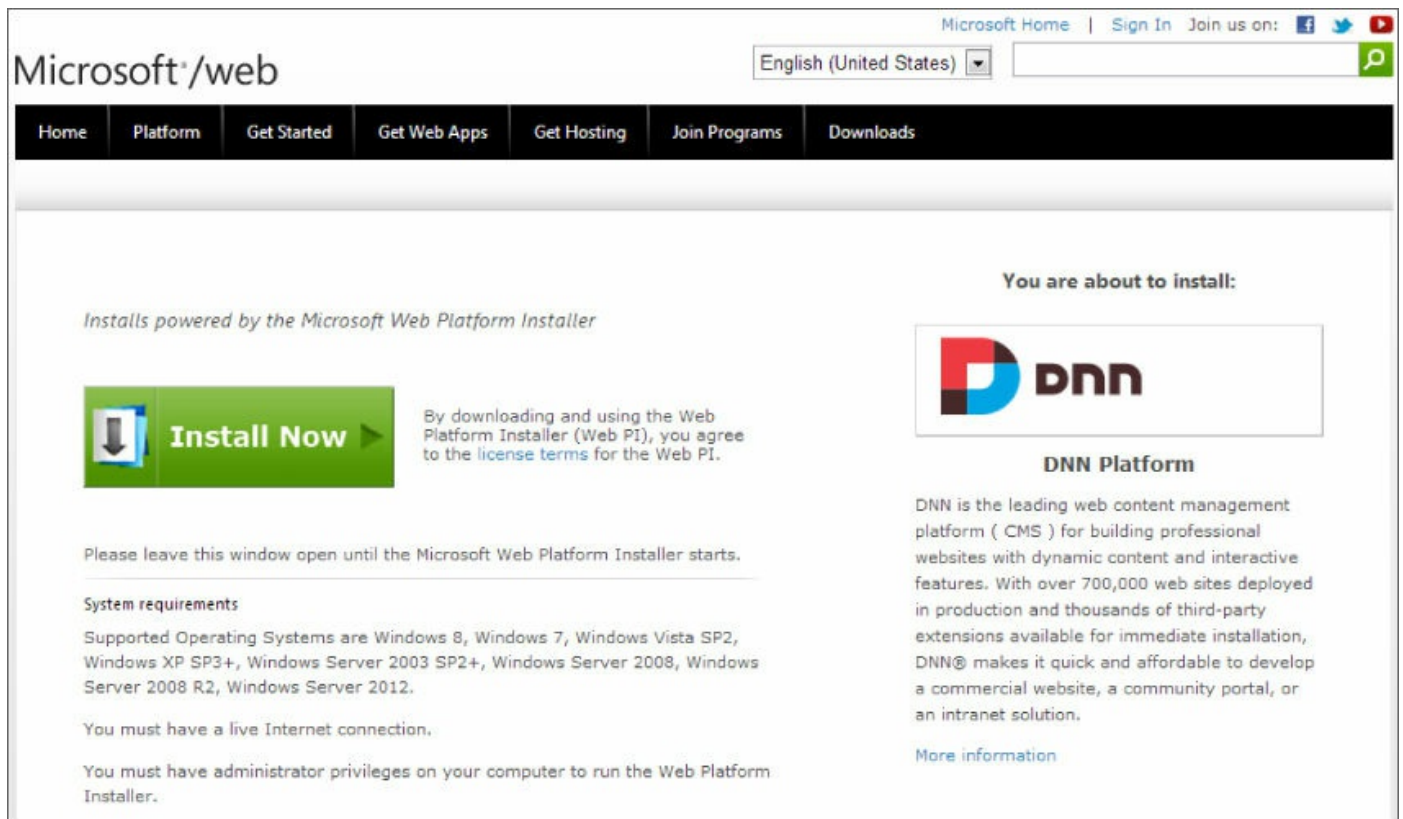
Installing the DNN Platform 7 using the Web Platform Installer is fairly easy. First you download the software, and then you install the DNN Platform and configure your website.

Downloading the Software

The Microsoft Web Platform Installer (Web PI) is a free tool that makes getting the latest components of the Microsoft Web Platform, including IIS, SQL Express, and every other component, easy. At the same time, the Web PI makes it easy for you to download and install free web applications like the DNN Platform.

On the DNN Software website you can find a link that points to the Web PI download and preselects the DNN Platform for installation. You can go to the

downloads section at <http://www.dnnsoftware.com/Community/Download> and click the Download DNN Platform button under the standard process. This takes you to the Microsoft website, as shown in [Figure 2.8](#).



[Figure 2.8](#)

On Microsoft's website click Install Now to start the download of dotnetnuke_iis.exe.

Installing the DNN Platform

In the previous step, you downloaded the Web PI from Microsoft's website. In this step, you're going to run the Web PI and install the DNN Platform.

To start, run the newly downloaded software to launch the Web PI with the DNN Platform, and click Install, as shown in [Figure 2.9](#).

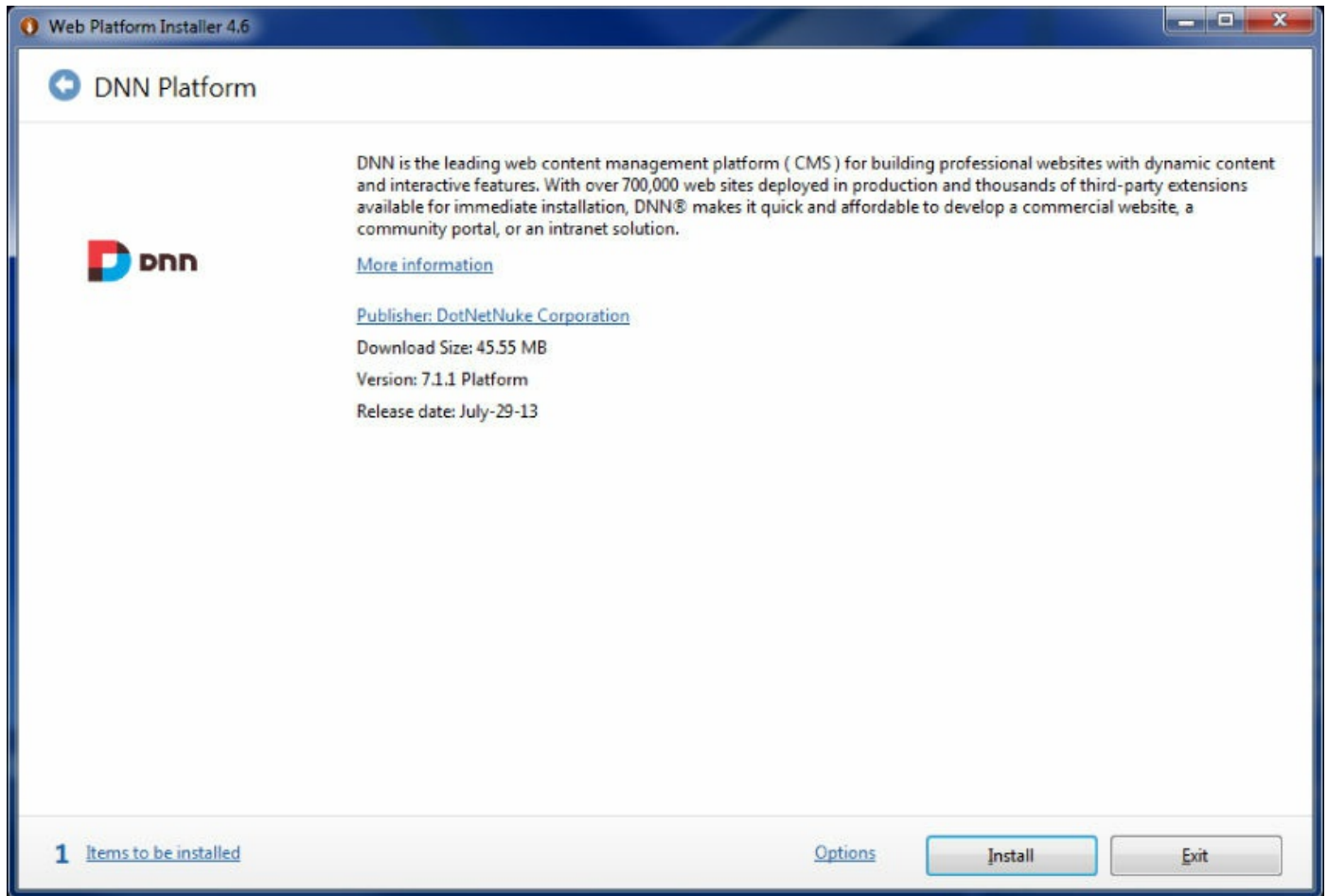


Figure 2.9

A pop-up with the list of prerequisites opens, as shown in [Figure 2.10](#).

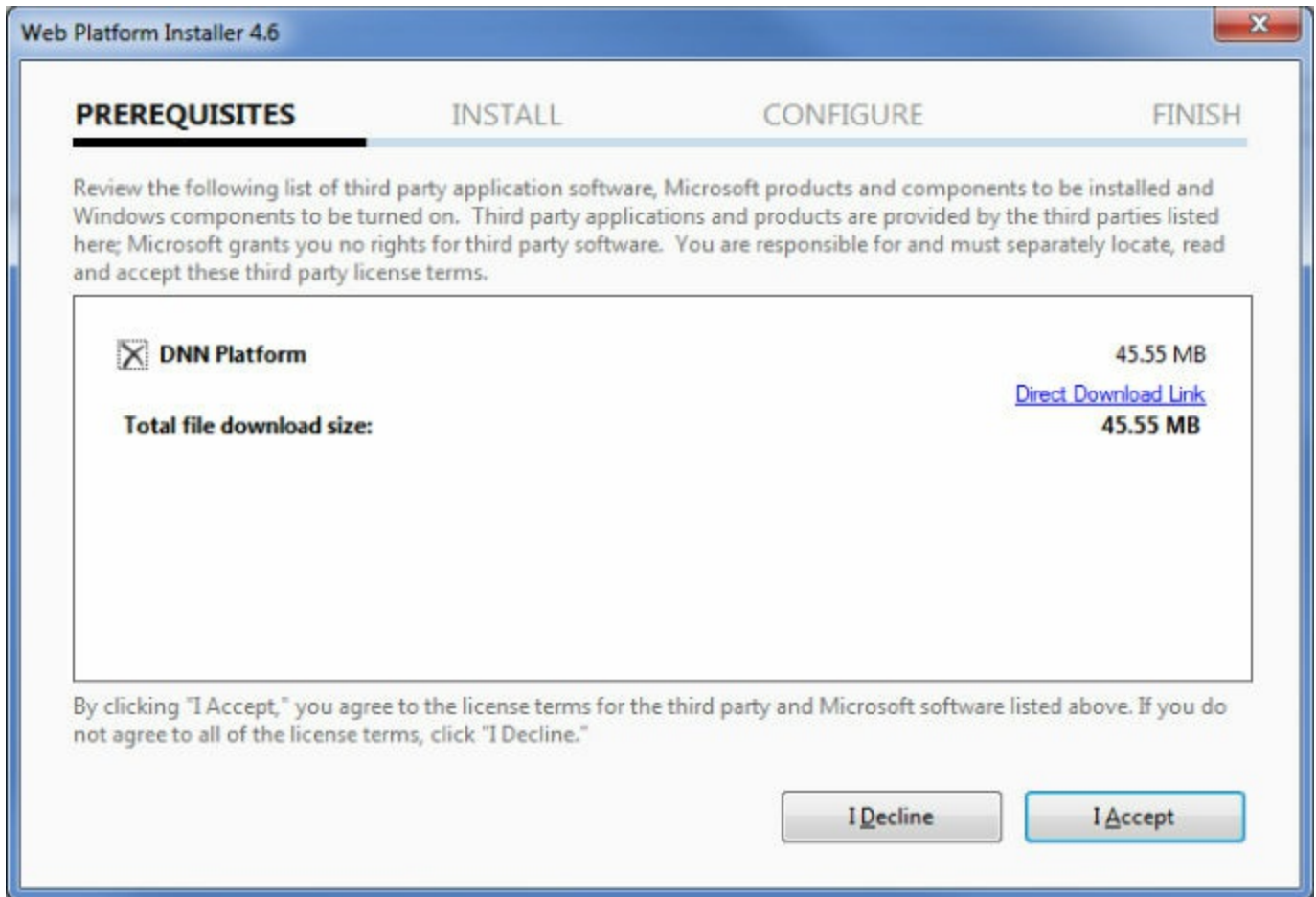
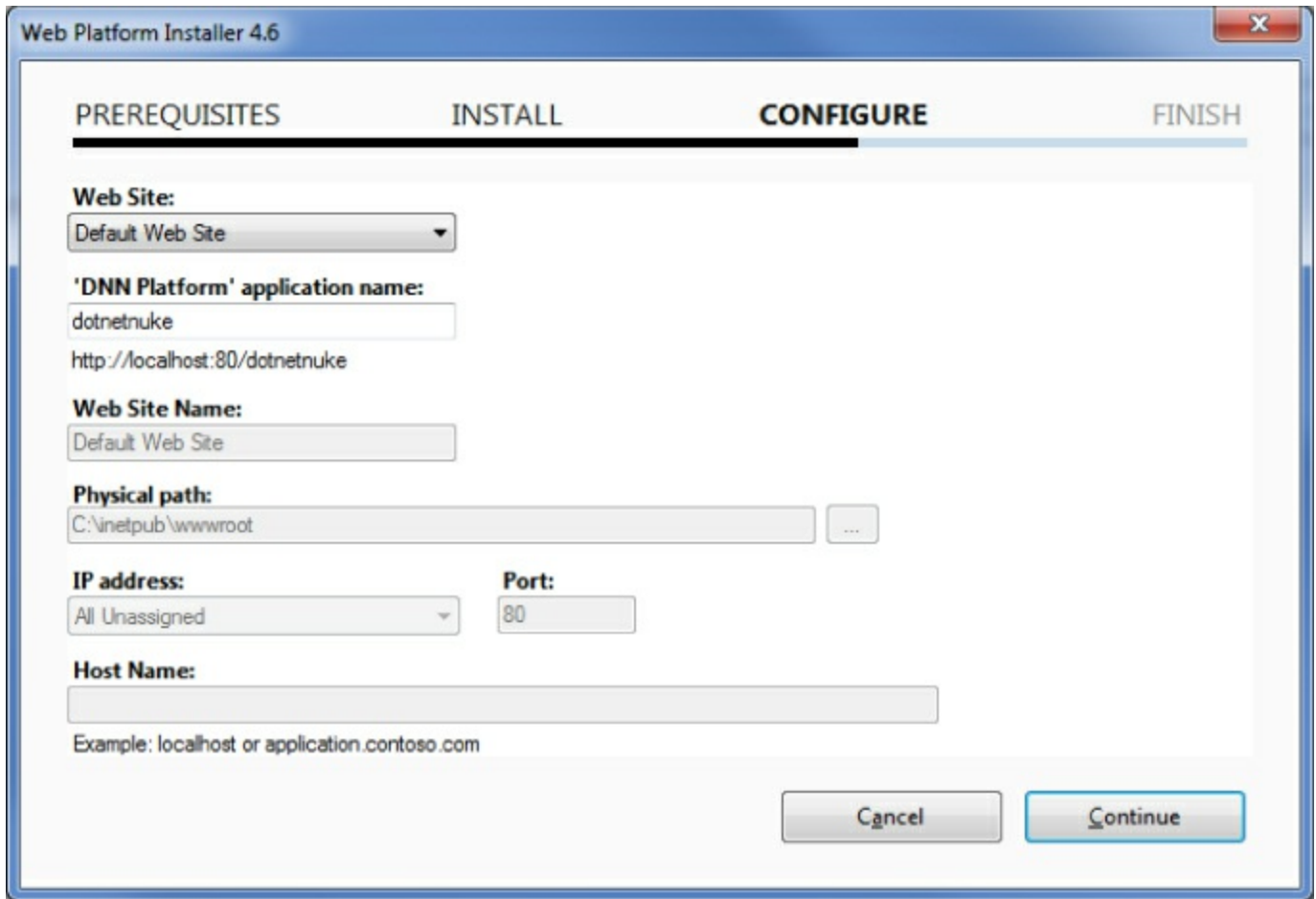


Figure 2.10

On this screen click I Accept, and the DNN Platform package, along with any prerequisites, will be installed. Then, a new pop-up appears where you can choose a name of your application, as shown in [Figure 2.11](#).



[Figure 2.11](#)

Click Continue to start the configuration of your website. You'll see a pop-up with a Finish button, as shown in [Figure 2.12](#).

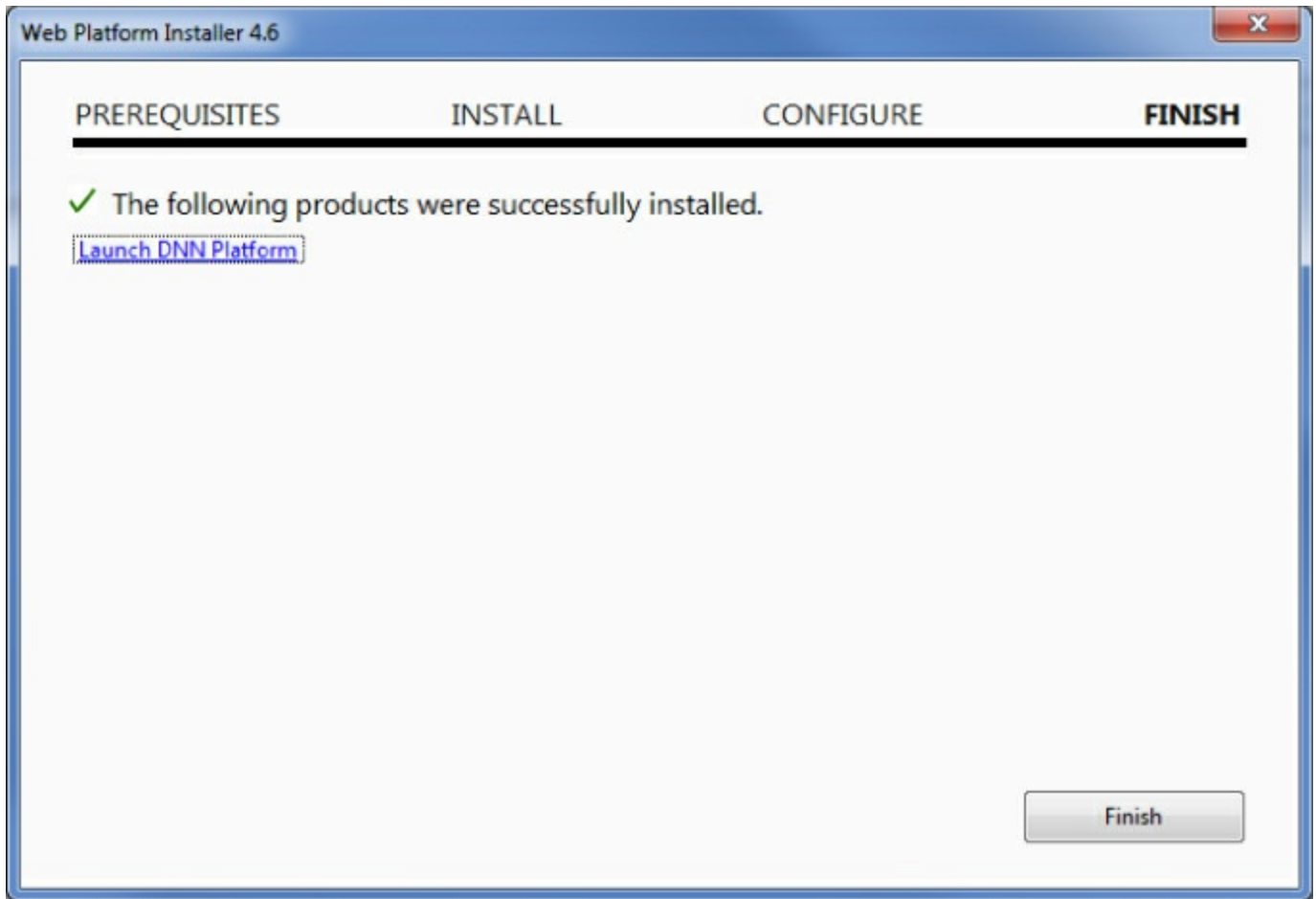


Figure 2.12

Click Finish and to launch the installation of your site as described in the “Step 7: Perform the Installation” section. Follow the directions in that section, and your site will be ready to go.

Upgrading the DNN Platform to Version 7

Upgrading the DNN Platform is even easier than installing it. The Upgrade Package contains the new files and a very simple flow that will guide you during the upgrade process. This section helps you upgrade from the DNN Platform version 6 to 7.

First, create a backup of your website. We recommend that you back up your website files along with your entire database. Always back up your site before any system changes to be able to recover from any possible errors.

Download the software from the DNN Software website (<http://www.dnnsoftware.com/Community/Download>). As described earlier in this chapter, go to the website, register or log in, and download the package under the upgrade section.

After you download the upgrade package, unzip it and copy the files over your existing installation. If Windows prompts you, select the Overwrite option.

Load your site by navigating to the URL. You'll see a screen that prompts you for a host username and password, as shown in [Figure 2.13](#). Type the credentials and click Upgrade Now.

DNN

US Germany Spain France Netherlands

Upgrade

Current Version - 06.02.09
Upgrade - Version 07.01.02

You are about to upgrade your website to a more recent version of the application. Applying upgrades on a consistent basis is the best way to ensure that you are protecting the integrity of your investment and the security of your users and assets. Before proceeding with the automated upgrade process please ensure that:

- you have made plans to first attempt this process in a staging environment
- you have documented your current installation characteristics including doing research on the compatibility of any third party modules which you may be using
- you have created the necessary backups of your environment so that you will be able to restore your website in the event of an unexpected upgrade failure.

1 Account Info 2 Upgrade View Website

Host (SuperUser) Username: *

Password: *

[Upgrade Now](#)

Figure 2.13

Your website starts updating, and you are redirected to a screen that shows the progress of the upgrade, as shown in [Figure 2.14](#).

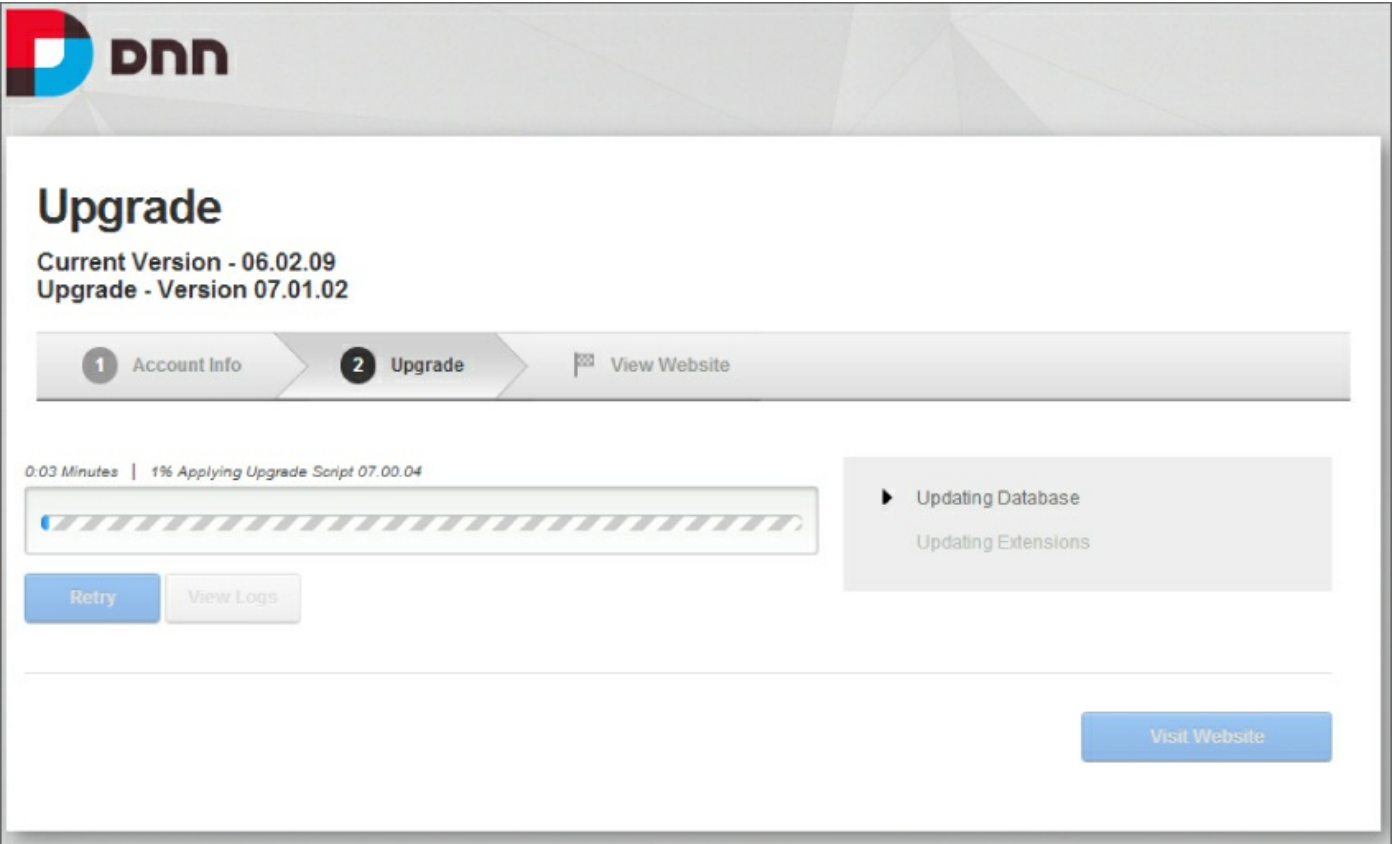


Figure 2.14

When the upgrade is complete, the progress bar shows 100%, and the View Website button will be enabled. Click it and your site will be ready.

Getting a Trial Version of Evoq Content

Evoq Content is a product built on top of the DNN Platform, and it is targeted to organizations with advanced content management needs. For more information about the features, see [Chapter 20](#).

In this section, you go through the process of getting a 30-day trial version of Evoq Content running on DNN's Cloud Environment. DNN's Cloud Trials offers the same infrastructure and reliability as production websites, and a trial version can evolve into the production website, which means that the time invested in testing the product can be used in the production website.

The process of getting your trial version takes just a few minutes. Navigate to the DNN Software website and log in or register if you don't have an account yet. After you log in, navigate to

<http://www.dnnsoftware.com/Solutions/Try-DNN> and click Get Started under the Evoq Content section. The trial registration form appears, and your information is prepopulated. Review that your information is correct, enter anything that may be missing, and click Start My Free Trial Now. The system will trigger the creation of a new instance of Evoq Content, and you'll be redirected to your site in just a few seconds.

Common Installation Issues

Installing the DNN Platform is a relatively simple process. Although very few things can go wrong, problems can appear at times. The two main causes for failure are

- An invalid connection string to the database, which means that the information for the database entered during installation is not correct. To fix the problem, verify the information and try again.
- Insufficient file permissions. This happens when the user under which the website is running does not have the appropriate permissions to the root folder of your website. Make sure you follow the instructions found earlier in this chapter for setting file permissions.

Finally, the DNN Platform has a large community of users. If you experience more problems, go to the DNN Software website and search for forums or enter questions to get additional help. One of the thousands of passionate users will be happy to assist you.

Summary

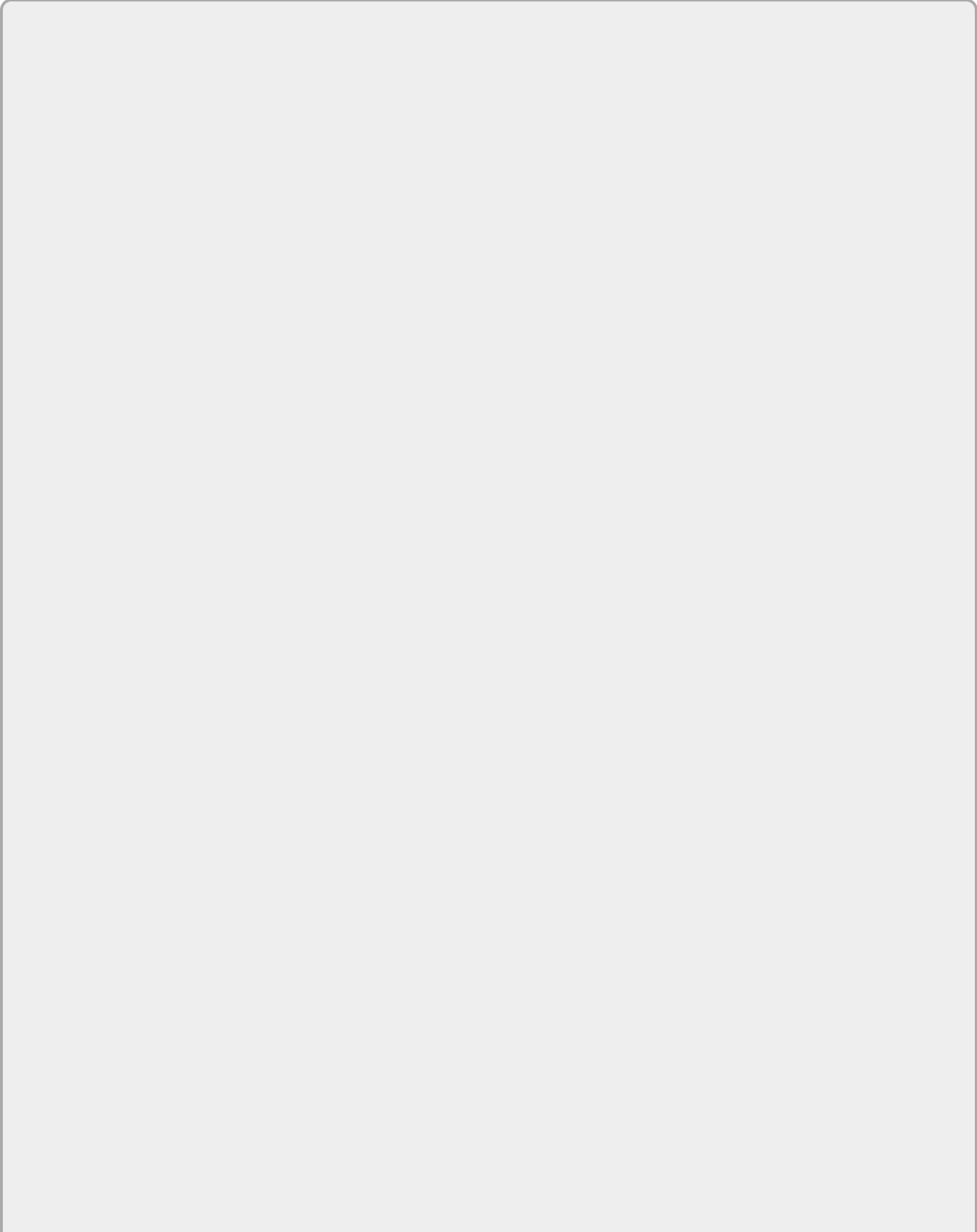
The DNN Platform has a few packages to choose from.

- The Install Package contains only the files needed to deploy a new site on a web server.
- The Source Package contains the application code including every module distributed with the Install Package.
- The Deploy Package contains the files needed to deploy a new site using Microsoft's Web Platform Installer.
- The Upgrade Package contains only the files needed to upgrade your existing website to the latest version.
- Evoq Content Trial is a cloud-hosted environment for you to test the advanced content management features built on top of the DNN Platform.

This chapter covered the different packages and the necessary steps to install a DNN Platform website.

Chapter 3

DNN Platform Overview



What You Will Learn in this Chapter

- Understanding the basic concepts of the DNN Framework
- Introduction to DNN security

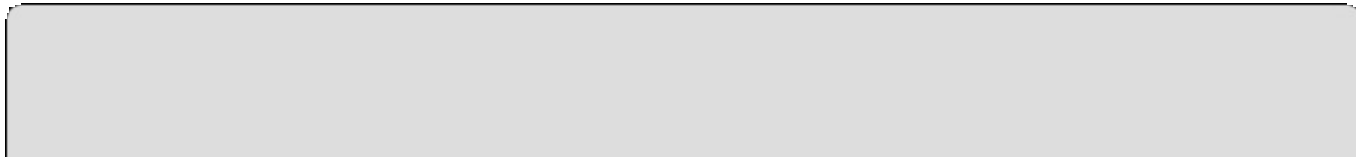
This chapter introduces you to some of the core concepts of the DNN Platform and provides an overview of how these fit together. Over the years, DNN has evolved and changed appearance considerably, but the overall structure has remained the same. What you have installed through the previous chapter is an extensible “web application framework.” You can use it to create all kinds of web-based applications. But to do so you need to understand some of its underpinning core concepts. And to complicate things a little bit, the terminology has evolved over the years as well. This means that some concepts have a different name in the UI than they have in code (and in SQL) purely because at one point it was decided that the old term was confusing or too “techie,” so it was dropped for something more easily understandable. Because this is a largely technical book, we will stick to the terms used in the DNN API, but we will note if they have another name in the UI.

Core Platform Objects

In this section, the basic building blocks of DNN and how they fit together are examined from a technical perspective.

Sites (Portals)

DNN is a powerful engine for creating websites. That is *websites*, plural and not singular. One of DNN's core features is what is known as “portal virtualization,” which means a single DNN *instance* can serve multiple discrete websites. Here, an instance is synonymous with a DNN *installation*. Here, an instance is synonymous with a DNN installation, meaning the files you've copied to your server's DNN folder with a web.config in the root.



TIP

A universal way to spot a .net web application is the web.config file in its root directory.

The aforementioned websites are known as *portals*. Portal is the DNN API name for what you call a *website* or simply a *site*. It stems from the very first version of DNN and an era where you had several popular Internet portals like Yahoo that aggregated content from other websites. The visual and textual language of DNN is rooted in this time, which is why the term *portal* is used to describe a site.

So, DNN can have multiple sites. Each site is accessible through a URL consisting of a domain (www.acme.com) and optionally a path. This URL is referred to as a *portal alias*. And in DNN a single site can have multiple aliases. So, a single site can appear as <http://www.acme.com> and <http://acme.com> as well as <http://public.acme.com>. Whenever a request is made to your server, IIS is responsible for routing the various URLs (or *domains* as they're often called) to the correct applications. Your server's administrator would be responsible for making sure all these URLs map to the DNN instance. After that, it's DNN that must decide which portal to serve based on the incoming URL. So, you can specify multiple aliases for each portal to let DNN route multiple URLs to the same website.

If you look at the page where you specify a new portal (you find this under Site Management on the Host menu), you notice that there are two types of portal (Site Type): parent and child. See [Figure 3.1](#). These terms can be somewhat confusing, as the child portal is not embedded within the parent portal at all. Instead, the only difference is in the form of the portal alias. Parent portals have at least one unique domain, so the URLs mentioned previously are all for parent portals. Child portals, on the other hand, are defined like subdirectories of a domain. So, a child portal given the previous installation could be <http://www.acme.com/public>. Now the “public” portal is defined as a child portal. Still, all its data is kept strictly separate from <http://www.acme.com> (if that were a parent portal on the same installation).

The screenshot shows a web browser window with the address bar displaying 'http://www.secondsite.dev'. The page title is 'Test DNN Site > Site Management > Add Site'. The form contains the following elements:

- Site Type:** Radio buttons for 'Parent' (selected) and 'Child'.
- Site Alias:** Text input field containing 'www.secondsite.dev'.
- Home Directory:** Text input field containing 'Portals/[PortalID]' and a 'Customize' button.
- Title:** Text input field containing 'A Second Website'.
- Description:** Text area.
- Keywords:** Text area.
- Template:** Dropdown menu showing 'Default Website - English (United States)'. Below it is a blue button labeled 'Default Website Template'.
- Use Current User as Administrator:** A checked checkbox.
- Buttons:** 'Create Site' (blue) and 'Cancel' (grey).

Figure 3.1

So, why would you use child portals? There are a couple of scenarios where this is useful. The first concerns cookies. Browsers typically protect the user's privacy by blocking any website from accessing cookies set by other websites. A cookie set by Google cannot be read by Apple and vice versa. You can see where I'm going with this. The child portal's cookies will be accessible by the parent portal and vice versa. Crucially, this makes it possible to keep a user logged in (login status relies on cookies). So if a user has been shared between a parent and a child portal, that user could navigate between the two sites and not have to log in every time he or she switches (so-called single sign-on). Another reason for using child portals is using them to separate departments of an organization. So, acme.com/dept1 is a different site from acme.com/dept2. This allows the departments to manage their own site with all the benefits that brings. A third reason to use child portals could be

one of access rights. For a new parent portal you need to add the new domain to IIS and to a DNS record. The person who is allowed to create new portals might very well not have that authority/access. In that case, a child portal is an alternative to create a new site without the necessity to access other company resources.

The data for each portal can be divided into database data and file data. Database data is stored in the various tables of your DNN database and file data, which is stored in some file-based storage location and called the *home directory*. By default, this is found in the directory portals/[PortalID] of your installation. You can specify another location on your server's hard drive. Since version 7.3 of DNN, you can specify alternate (cloud-based) locations for this storage. (You can even extend the framework and build your own provider for this.) The point of this paragraph is that you realize that the files for each portal are stored in discrete folders somewhere. And DNN makes sure that no files “leak” from one portal to another. Your portals are thus really discrete entities inside the framework. This is crucial if you think of the following use case. As a successful developer, you have created a DNN installation that is perfectly tuned to serve small businesses in your local area with a website. For each customer you create a new portal:

<http://www.joeplumbing.com>, <http://www.jilldrycleaning.com>, and so on.

But you need to have peace of mind that the content of these sites is secured from each other. Knowing how DNN organizes content gives you this peace of mind.

Finally, you should know that each portal has a unique *portal ID* and *GUID*. The GUID (Globally Unique Identifier) is a relatively new addition, but it's the portal ID that is used throughout the framework to identify a portal. Numerous tables in SQL have a column portal ID that refers back to the portal (stored in the Portals table). By default, the portal ID is used in the creation of the subdirectory under Portals for the Portal home directory. See [Figure 3.2](#).

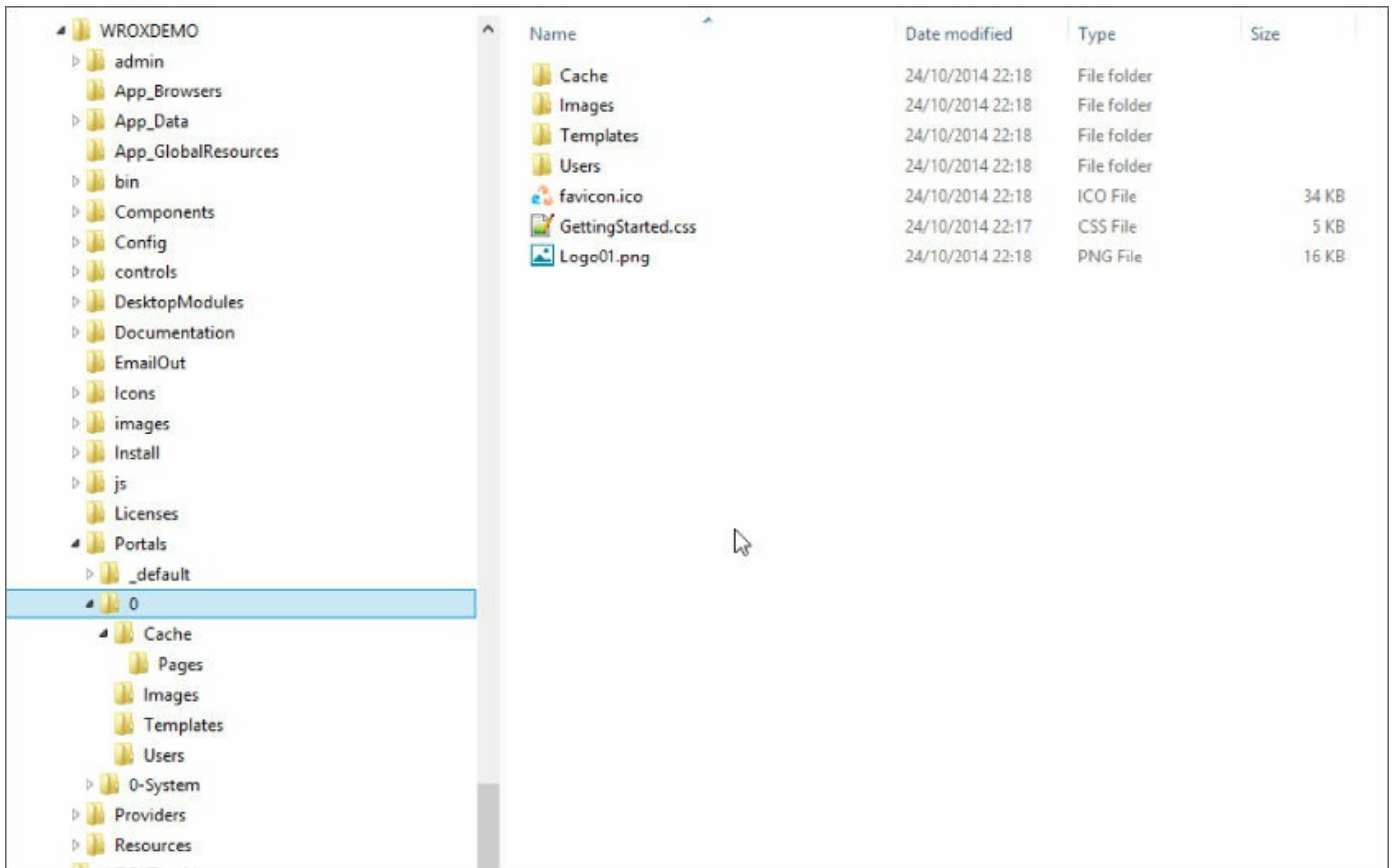


Figure 3.2

Pages (Tabs)

Each website consists of a number of pages. Or to put it in DNN parlance: each portal has a number of *tabs*. The word *tab* stems from the project from which DNN was originally derived: IBuySpy. In that sample project, Microsoft demonstrated how you could build a website using a tabbed layout. The term for page is thus *tab* in the DNN API. Back in the bad old days of static websites (before the widescale adoption of CMSs), the pages of a website were all discrete files (often HTML files) on your web server. So, there typically would be a file `index.html` that would be served by the web server when the browser was first directed at the site. Then any page would simply point to a file on the web server. We parted with this way of doing websites after the introduction of web applications like DNN. Instead, the HTML that is served to the browser is built up on the fly by the web application on the server and sent immediately to the client. No HTML file exists any longer that stores that HTML that was served. Instead, the application typically uses some form of templates that are subsequently filled with data from a database. In this way we can show a “Hello Peter” page when Peter is logged

in and a “Hello Shaun” page when Shaun is logged in.

The various tabs are stored in the Tabs table in the SQL database. And just like portals, tabs have a unique *tab ID* (an auto-incrementing integer) and GUID (called UniqueID in the table), but the tab ID is used throughout the framework to identify them. See [Figure 3.3](#).

TabID	TabOrder	PortalID	TabName	IsVisible	ParentID	IconFile	Di
7	1	NULL	Host	False	NULL		Fal
16	1	NULL	Host Settings	True	7	~/Icons/Sigma...	Fal
17	3	NULL	Site Manageme...	True	7	~/Icons/Sigma...	Fal
20	7	NULL	Vendors	True	7	~/Icons/Sigma...	Fal
21	9	NULL	SQL	True	7	~/Icons/Sigma...	Fal
25	11	NULL	Schedule	True	7	~/Icons/Sigma...	Fal
33	13	NULL	Lists	True	7	~/Icons/Sigma...	Fal
34	15	NULL	Superuser Acco...	True	7	~/Icons/Sigma...	Fal
36	17	NULL	Extensions	True	7	~/Icons/Sigma...	Fal
37	19	NULL	Dashboard	True	7	~/Icons/Sigma...	Fal
40	23	NULL	Configuration ...	True	7	~/Icons/Sigma...	Fal
55	1	0	Home	True	NULL	NULL	Fal
56	3	0	Module	True	NULL	NULL	Fal
57	9	0	Activity Feed	False	NULL	NULL	Fal
58	11	0	Admin	False	NULL	NULL	Fal
59	13	0	Search Results	False	NULL	NULL	Fal
60	1	0	My Profile	True	57	NULL	Fal
61	3	0	Friends	True	57	NULL	Fal
62	5	0	Messages	True	57	NULL	Fal
63	1	0	Advanced Setti...	True	58	~/Icons/Sigma...	Fal
64	3	0	Site Settings	True	58	~/Icons/Sigma...	Fal

Figure 3.3

A web application like DNN includes logic to generate and interpret URLs because the pages are no longer files on the drive. This is not a trivial matter. URLs are crucial to SEO, for example, so you'll find many are very opinionated about this aspect of the framework. To cater for any scenario, DNN has detached this part of the application so anyone can write his or her own version. The generic term for this is *URL rewriter*. A URL rewriter constructs URLs that are displayed in the application that can then be subsequently interpreted by it and mapped to a specific tab ID. The first versions of DNN did not include much URL rewriting, and you would have seen URLs like this: <http://www.acme.com?tabid=32>. As you can probably guess, this URL would show the tab with ID 32. But in essence all URL

rewriters that exist even today just translate the incoming URL (for example, <http://www.acme.com/products/dynamite>) to something similar to that raw URL with `tabid=xyz` that is then used internally to find the right tab.

Pages are organized in a tree. See [Figure 3.4](#). That is, they can be nested. In the Tabs table the ParentID value points to the TabID value of the parent tab. This is how most modern websites are currently structured. Any page sits somewhere in a tree hierarchy. And ideally the URL rewriter uses this hierarchy to create “human-friendly” URLs that reflect this by using the titles as if they were subdirectories. The aforementioned URL would point to a page with the title “dynamite” underneath the page with the title “products.” Not only does this make sense to humans, but it's greatly rewarded by Google in your ranking.

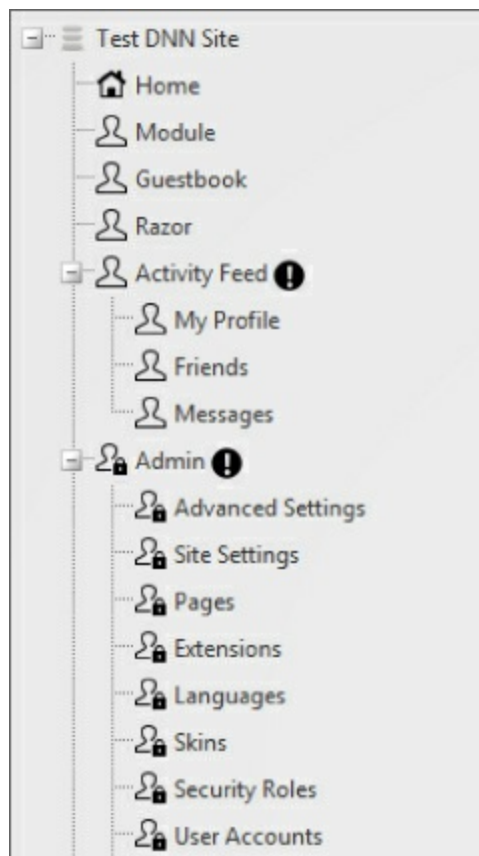


Figure 3.4

Theme (Skin)

Every page tells DNN which theme (until version 7.4 the term *skin* was used for theme, and you'll still find it in various places in the API) to load to display it. The theme defines how various elements are positioned on the page and how they look. The DNN Platform aims to separate the concerns of

programmers, designers, and content managers as much as possible. So, designers should be able to do their jobs independently from programmers, for example. What this means is that you can change the look and feel of a site without changing its functionality or content.

A theme is typically comprised of some HTML template file (.ascx), JavaScript files, and CSS files. And a single theme package can contain multiples of these theme templates. Often you'll find a template for the home page, for an admin page, and for a regular (default) page. The home page would then include some extra space for a banner/image slider, for example. In the site settings you can specify the default skin to use for the site, but any page can override this setting and specify its own special skin to use.

The most important element on a page is undoubtedly the one responsible for navigation—the menu. Other common elements are the login button, the logo, and the banner for the site, as well as the footer elements such as copyright messages and disclaimers. Obviously, DNN also needs to know where to put the page's contents. For this, the skin template defines a set of *panes*. Panes are essentially div elements that have been given meaningful names by the designer: LeftPane, RightPane, BannerPane, and so on. See [Figure 3.5](#). These names appear in the UI and allow the user to specify where something will be added on the page. If you switch to layout mode (use the top-right menu on the control bar), you'll see all these panes light up, and you will see how the designer has subdivided the page for you.

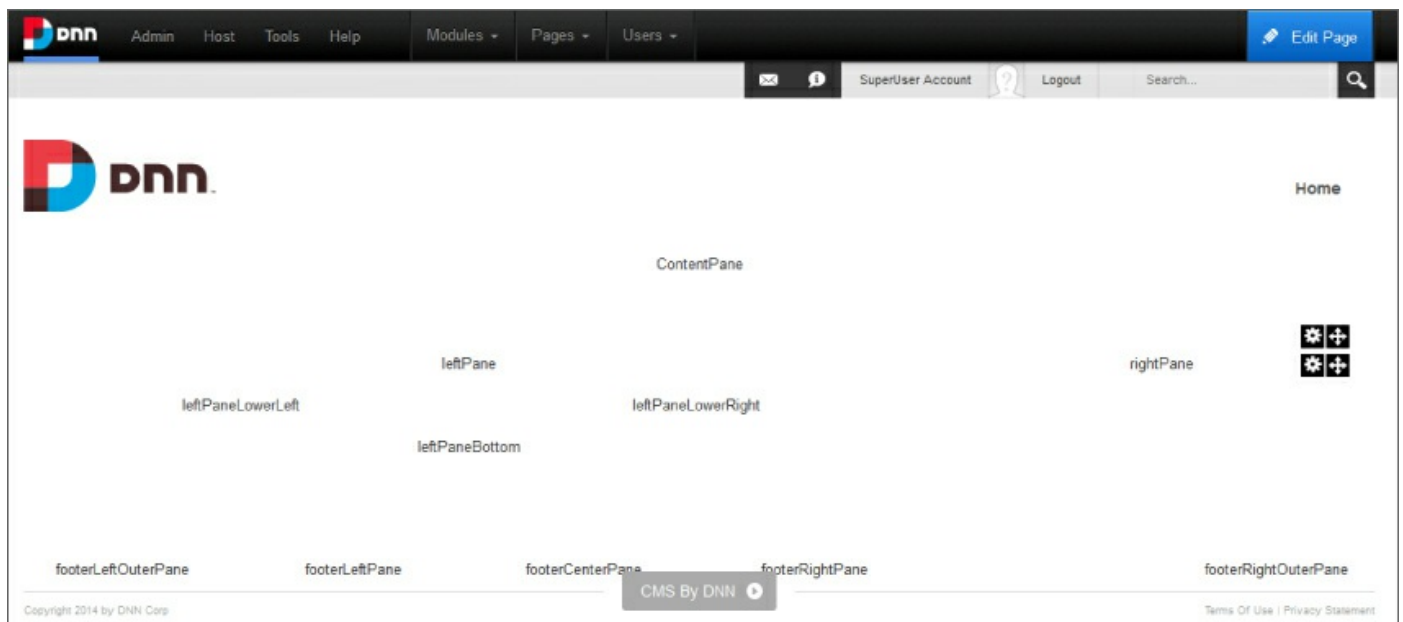


Figure 3.5

Modules

Each tab in DNN displays any number of *modules*. Modules are basically the rectangular blobs you see spread on the page's surface. Each one has a small menu when you switch to edit mode (to switch the mode, go to the top-right side of the page on the control bar). See [Figure 3.6](#).

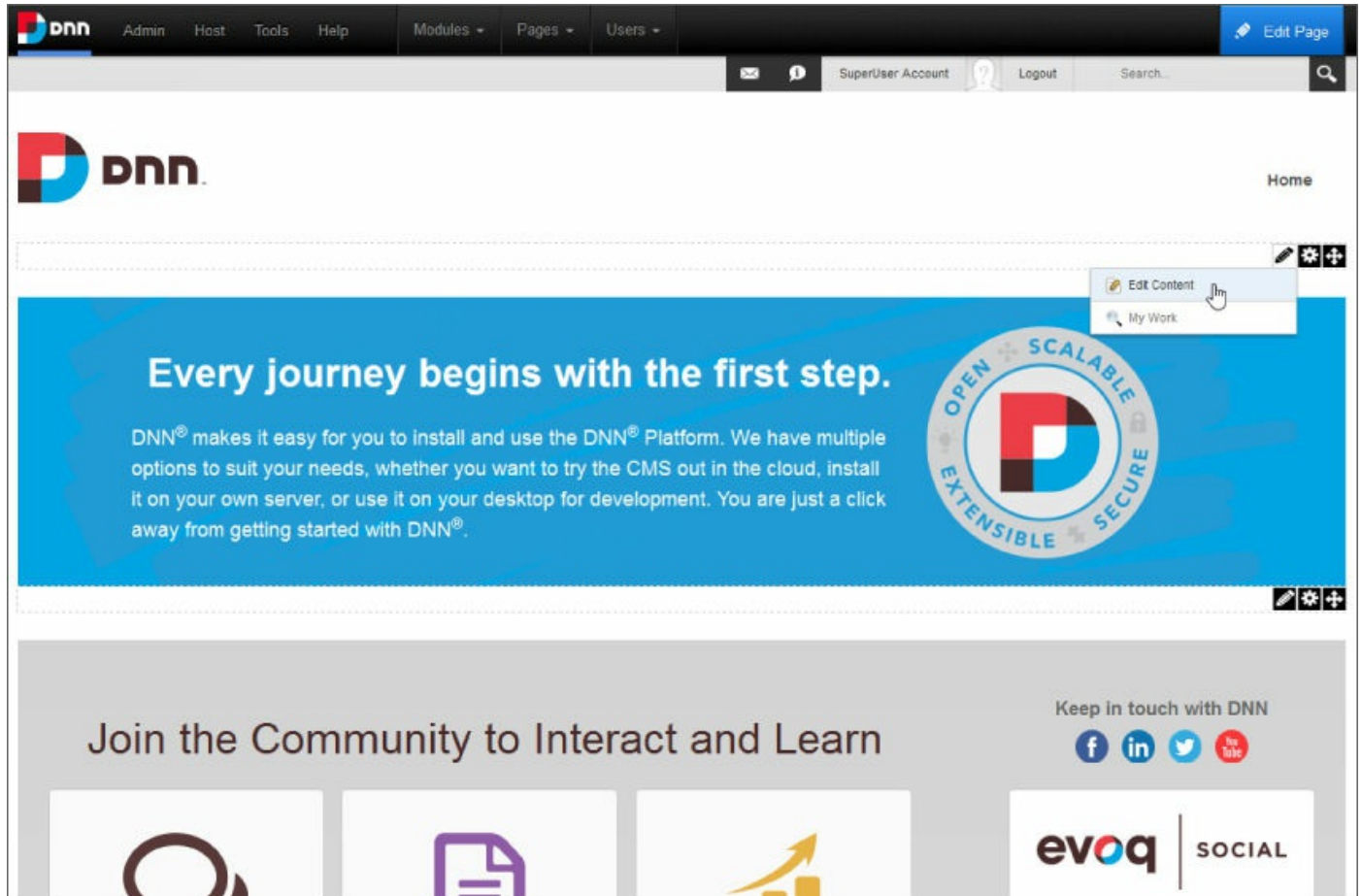
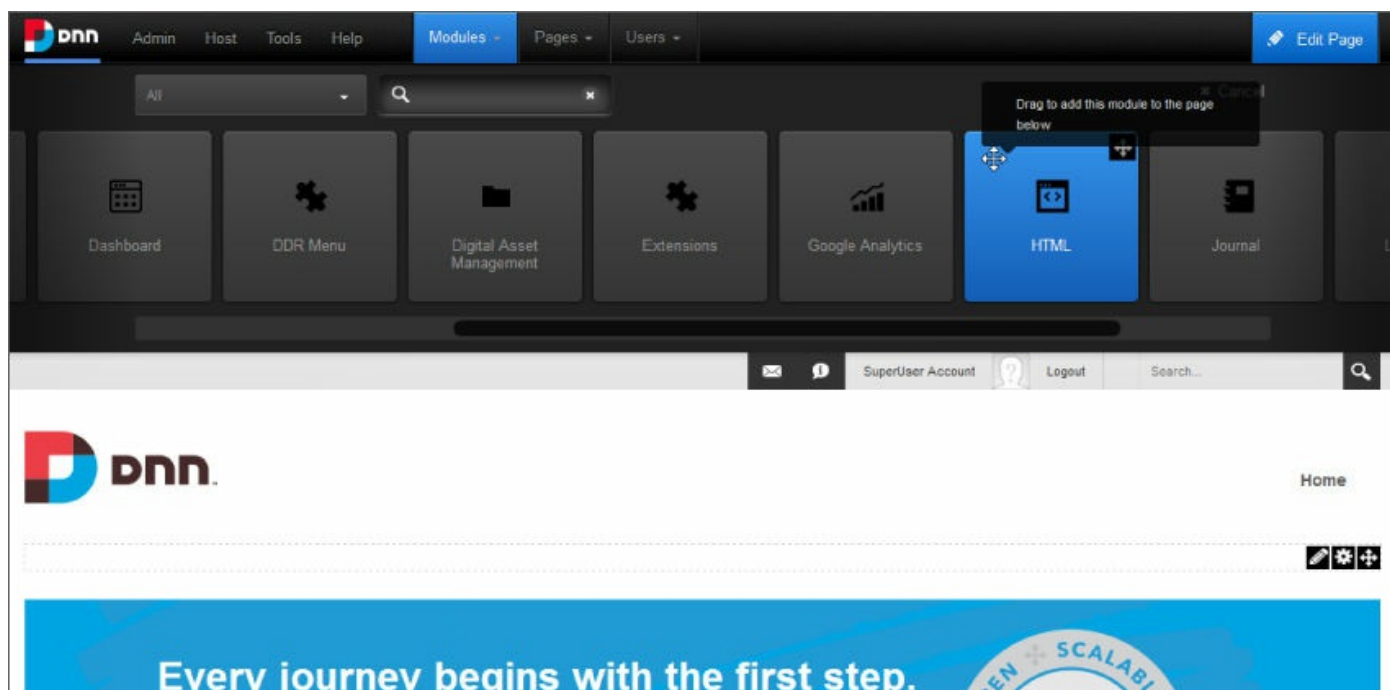


Figure 3.6

These modules are what make the site tick. They are the fundamental building blocks of functionality of your site. You can have a module that displays a block of text (the most common module by far), but you can also have a module that shows a small calendar with events or a module that displays a list of (recent) news articles. And it all began with what has been dubbed “Web 2.0.” That is, the web is no longer a collection of static text and images but rather an interactive surface that allows people to modify its contents. Modules are the “apps” of the web. They perform a particular function (for example, manage an event calendar). And it is one of the most important extension points of DNN: you can create your own modules (explained elsewhere in this book) and let the users stick that on any page of

their site. And the ease with which this can be done and the power it provides the developer is the top reason for DNN's success.

Modules are instantiated on a page through the Modules tab on the control panel. See [Figure 3.7](#). If you click Add Module, a list of installed modules appears, and you can drag and drop them onto your page surface. Typically, you then set some parameters such as who can edit its contents and some generic parameters that concern the specifics of the module (for example, whether to render a calendar or a chronological list of events). This is commonly referred to as the *Module Settings* and can be accessed through the little menu that appears when you are in edit mode (see [Figure 3.6](#)).



[Figure 3.7](#)

The installed modules on your DNN installation are managed on the Extensions page on the Host menu. Here you see what is installed and what is ready to install (has already been downloaded), and you can access the DNN Store for more modules (both free and commercial). See [Figure 3.8](#).

Host > Extensions

Extensions

[Install Extension Wizard](#)
[Create New Extension](#)
[Create New Module](#)

[Installed Extensions](#)
[Available Extensions](#)
[Purchased Extensions](#)
[More Extensions](#)

Expand All

This application contains an Update Service which displays an icon when a new version of an Extension is available. The icon displayed will contain a visual indication if a currently installed Extension contains a potential security vulnerability. If a security vulnerability is identified, it is highly recommended that you upgrade to the newer version of the Extension. Clicking the icon will redirect you to a location where you will be able to acquire the Extension for immediate installation.

Modules





















	Name	Description	Version	In Use	Upgrade?	
	AdvancedSettings		1.0.0	Yes		
	Authentication	Allows you to manage authentication settings for sites using Windows Authentication.	7.1.0	No		
	Banners	Banner advertising is managed through the Vendors module in the Admin tab. You can select the number of banners to display as well as the banner type.	7.1.0	No		 
	Configuration Manager		7.1.0	Yes		
	Console	Display children pages as icon links for navigation.	7.1.0	Yes		
	ContentList	This module displays a list of content by tag.	7.1.0	No		
	Dashboard	Provides a snapshot of your DotNetNuke Application.	7.1.0	Yes		
	DDR Menu	DotNetNuke Navigation Provider.	7.3.4	No		 
	Device Preview Management	The Device Preview Management module allows users to manage their mobile preview profiles.	7.3.4	Yes		

Figure 3.8

The first time you see the list of installed modules, it may seem overwhelming. But most of those are administrative modules that you cannot (and should not) delete. They are part of the framework. Click the Install Extension Wizard button at the top of the page to also install modules that you've downloaded (or created) and bypass the DNN Store. This is not a closed system like most mobile platforms, and you'll find hundreds of free (and often open source) modules on sites like CodePlex and GitHub. A module is delivered as a Zip file. All you need to do is upload the Zip file through the Extensions page, and DNN takes care of the rest. This process has been made as easy as possible for both the administrator and the developer.

Even upgrades are taken care of automatically. Just upload the Zip file you've downloaded, and the module appears on the control bar so it can be dropped onto a page.

When a module is added to a pane, it is wrapped inside what is known as a *container*. This is similar to a skin in that it is defined by the designer and is a template file that tells DNN how the module is wrapped. The container typically determines how the title of the module is rendered. It can be used to draw a border around a module or give it an alternate background color.

Containers are typically distributed along with the skin, and just like the skin, you have a default container for the site and the ability to override the default for each module.

Security

Security is an integral part of web applications, and DNN has many features built in to it that allow you to create advanced applications. Module developers can leverage this and not be concerned with authentication and/or authorization at all. Instead, they can leave this to the framework and focus on their application. In turn, DNN devolves this to components that can be switched out. This is a very powerful feature that allows you to use Active Directory authentication or to leverage OAuth providers like Facebook.

Users

It should come as no surprise that *users* are individual login accounts in DNN. You find them in the Users table in SQL and on the User Accounts page on the Admin menu. See [Figure 3.9](#). Each user is uniquely identified through a username for authentication purposes. In data, however, users are identified through an auto-incrementing integer user ID similar to tabs and portals. Whenever a user account is created, that user will only be able to log in to the portal in which it was created. But the user can be shared between portals (UserPortals table in SQL) although there is no proper UI for that right now.

Admin > User Accounts

User Accounts

Search: Username

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 Online Unauthorized Deleted

			Username	Display Name	Address	Telephone	Created Date	Authorized
			aculverhouse79	Albrecht Culverhouse			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			admin	Administrator Account			11/14/2014 14:00:26	<input checked="" type="checkbox"/>
			asokyrko42	Andria Sokyrko			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			bdares12	Basheer Dares			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			bnomura48	Bart Nomura			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			ckingan46	Claire Kingan			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			csydoryk96	Claudelle Sydoryk			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			dgodllington90	Denna Godllington			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			dsammon31	Domini Sammon			11/14/2014 15:08:04	<input checked="" type="checkbox"/>
			dsloan81	Dutch Sloan			11/14/2014 15:08:04	<input checked="" type="checkbox"/>

1 2 3 4 5 Items per page: 10 Page 1 of 5, users 1 to 10 of 41

Figure 3.9

Roles

Roles are what drive security in the platform. See [Figure 3.10](#). A user can belong to one or more roles, and these roles are used throughout the application to set permissions. This is similar to what you'd see on your Windows machine. There are several roles that are installed by default: Administrators, Registered Users, Subscribers, and Unverified Users. Administrators have access to all aspects of a portal. Registered Users is a role that is assigned automatically to any new account. Subscribers is a role to which users can opt in voluntarily. Finally, Unverified Users is used for those users who have enlisted but have not yet verified (through an email link) that they are who they say they are. The thinking behind this last role is that you can accept a registration but exclude those from certain functionalities, as the registration would have already given that user the Registered Users role.

	Name	Description	Fee	Every	Period	Trial	Every	Period	Public	Auto	Users
	Administrators	Administrators of this Website	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	1
	Registered Users	Registered Users	0.00			0.00			<input type="checkbox"/>	<input checked="" type="checkbox"/>	41
	Subscribers	A public role for site subscriptions	0.00			0.00			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	41
	Translator (en-US)	A role for English (United States) translators	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	Unverified Users	Unverified Users	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0

Figure 3.10

These roles are specific to the portal. They have a unique role ID (again an auto-incremented integer) that is used to refer to them throughout the framework. There are two virtual roles that are worth mentioning here. They are virtual in the sense that they do not exist in the Roles table in SQL but are used in the API for specific cases. These virtual roles are All Users and Unauthenticated Users. The first (RoleID = -1) you will use to allow both authenticated and unauthenticated users access to something. The second is to target unauthenticated users only. So, you could show a login panel or some explanatory text to those users only, for example.

You see the roles come back whenever you edit permissions for specific part of the website. Both pages and modules on those pages have permissions.

[Figure 3.11](#) is a screenshot from a module's permission screen. You can add roles and/or users to the grid and set permissions by checking the boxes. What these permissions do depends very much on the module and is discussed in later chapters.

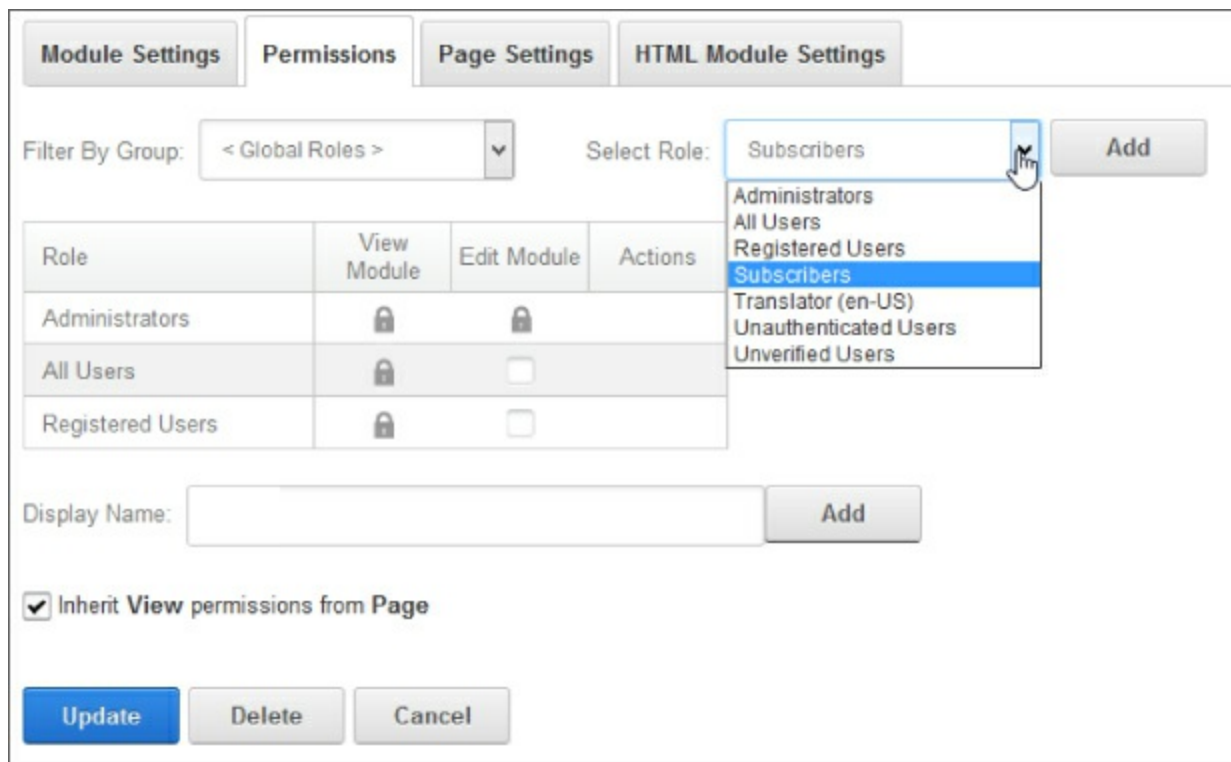


Figure 3.11

Host

Upon installation, DNN will create a first user account that is the *host* account. This account has administrative privileges in all portals as well as the ability to access all resources that are common to all portals. *Host* is synonymous with *superuser* in DNN. You can add as many host accounts as you want. These accounts are the only ones able to create and delete portals, install and delete modules, and so on. You will immediately notice you're logged in as superuser by the appearance of the Host menu in the control bar.

Admin

Admins are regular user accounts that have been added to the Administrators role. Admins will see the Admin menu on the control bar and have the ability to change the portal, such as adding and removing pages, creating and deleting users, creating and deleting roles, and so on. Admins cannot install

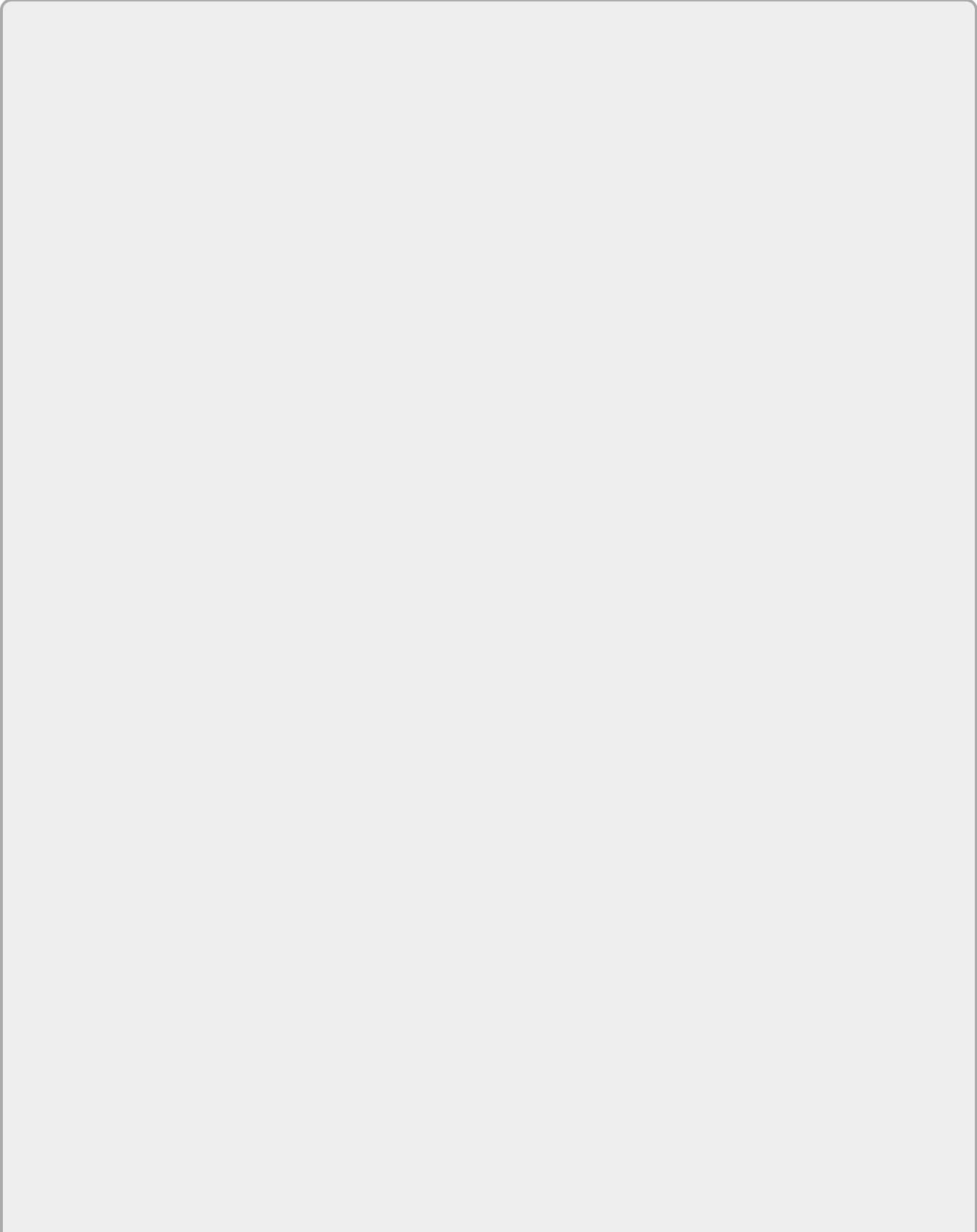
components to the system such as modules, nor can they alter files that run code on the server. Care has been taken to make sure that admins are sandboxed to allow them only the ability to change the content and appearance of a portal.

Summary

In this chapter, you looked at DNN from 30,000 feet high. You saw the hierarchy of the DNN instance, portals/sites, tabs/pages, and finally modules. You now have a rough idea of where your data is stored and that DNN uses this to construct the pages that are being served to the client's browser. You also learned how DNN organizes security using users and roles. Understanding this chapter is key to being able to comprehend more in depth how DNN works and how you can create your own website or web application with it.

Chapter 4

Site Administration



What You Will Learn in This Chapter

- Exploring administration features
- Taking first-time steps to set up your site
- Delegating administration duties
- Delegating content editing tasks

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 4](#) download and individually named according to the names throughout the chapter.

In the previous chapter, you learned what DNN is and what it can do for you. This and subsequent chapters will continue to build on that knowledge to ensure that you're the most productive and knowledgeable DNN user possible.

Starting with this chapter, we will teach you what you need to know in order to efficiently run your DNN site using the best practices we have learned over a decade of managing DNN sites over a multitude of use cases and deployment scenarios.

What Is Site Administration?

Site administration in DNN is quite simply the process, tasks, and features that equate to “running” a DNN website. This could include managing end users, security, site settings, configuration, and even managing content. However, for the purposes of this chapter, we will primarily focus on the Admin menu that's found in the Control Panel feature of DNN.

Site administration can take on many forms, but there are two use cases that you should be familiar with. First, there's site administration as defined by DNN. You enable this by adding the Administrators security role as one of the assigned roles for the account. That way, all “admin” features will be available for that user across that specific site.



NOTE

Security roles are being discussed at a very high and conceptual level right now but will be discussed in greater detail later in this chapter.

Site administration can take on a hybrid of another sort as well, but this is not defined by DNN. Instead, you as the site administrator can define a different kind of administration for your site. This would be a kind of administration where you define a subset of administration capabilities and assign them to more privileged users to avoid having to make them an actual administrator.

As an example, there may be times when you want to create a security role that might be named something like Site Admin. You would then create a page or pages on the site that contain some of the Admin modules, assigning permission to the newly created Site Admin security role so that only participants of that security role can see them. In these areas, you can also add references or links to features that normally require Administrator permissions, provided you set up your permissions properly in those other areas and that the chosen feature(s) allows this.

In summary, you can perform all of the configuration and content-management tasks that you need to in order to run your own site. This does not include installation of extensions or high-level configuration options such as the ones you will learn about in [Chapter 5](#).

Common Administrative Tasks

As an administrator in DNN, there are a great number of things that you are able to do. Since there could be a book written on that topic alone, this chapter focuses on the most common and useful things that you need to know in order to get started and be productive.

In many cases, there are differences between the DNN Platform and other commercial versions of DNN, such as Evoq Content or Evoq Social. When such differentiators are present, they will be mentioned. If you want to understand all of the features that make the commercial editions of DNN different, be sure to read [Chapters 20](#) and [21](#).

Control Panel Sections

When managing your DNN site as an administrator, you will most likely be doing most of your daily work in the Control Panel. As an administrator, you will be able to use nearly every part of the Control Panel. The only part missing is the Host menu, which is covered in [Chapter 5](#).



TIP

You do not need to actually click any of the menu sections in the Control Panel. Simply hover over a menu item to expand that section of the Control Panel if it is a menu.

Those with content management permissions in DNN are often referred to as “content editors.” Content editors can typically view only some of the Control Panel—the parts that are required in order to perform other related tasks, such as adding a module to a page.

It is worth noting that the Control Panel you'll be using is the one that ships “out of the box” with DNN. However, it is one of numerous extension points in DNN. Like modules, skins, and other extensions, the Control Panel can be replaced with a custom or third-party alternative.

Admin

The Admin section of the Control Panel is actually a menu that will drop down when your mouse hovers over it. If you click it, you will be brought to a page that lists all of the Admin pages similarly to how the admin pages were displayed in earlier versions of DNN. Don't get used to clicking this menu item to see the contents of the Admin menu, because the click-through might not be retained in future versions of DNN.

The Admin menu will list any pages that appear in the hierarchy of the site, but under the page named “Admin.” If you add your page as a child to the Admin page, it will appear in the Admin menu, in the Advanced Settings section in alphabetical order.

The Common Settings section first appears when the Admin menu drops down. It contains all the common menu options that site administrators need to access on a regular basis. Assuming you didn't click the menu itself, this means that these options are literally one click away. [Figure 4.1](#) shows the Admin menu portion of the Control Panel; it's specifically showing the Common Settings section.

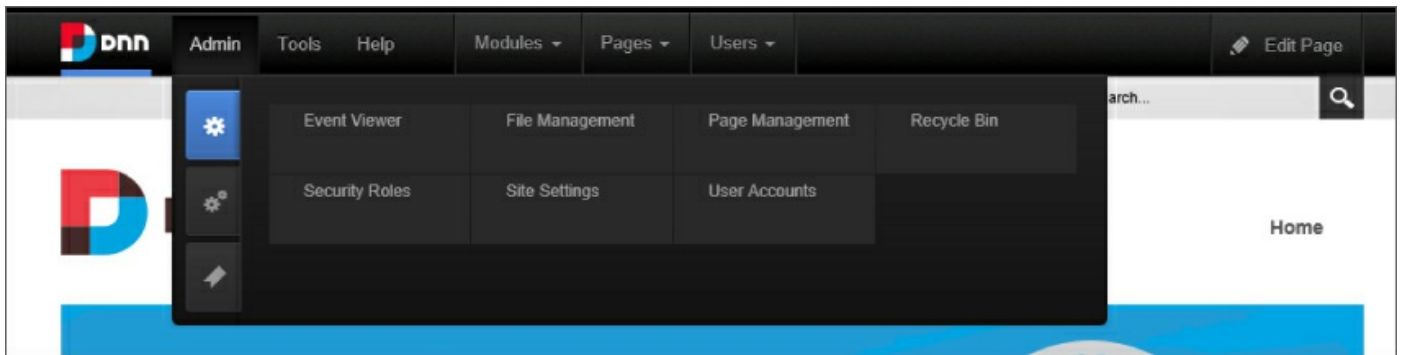


Figure 4.1

The Advanced Settings portion of the Admin menu (see [Figure 4.2](#)) is the next tab below the blue one. The “Advanced Settings” label might be deceiving to some people. The menu options in this portion of the menu are not really advanced at all. They’re just less commonly required. They’re all features of DNN that you don’t need to access very often or are set-it-and-forget-it kind of features.

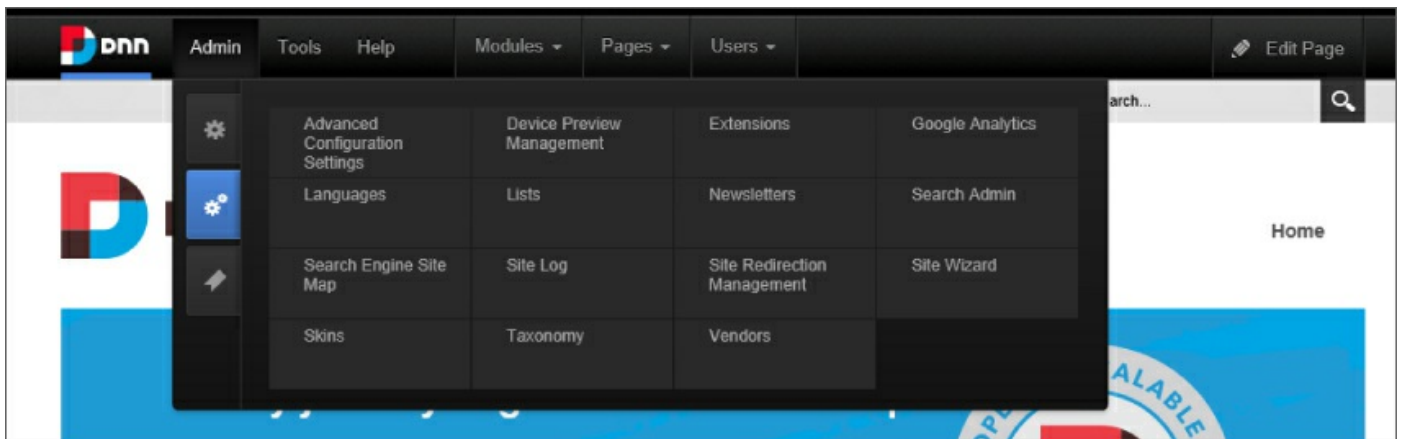


Figure 4.2


There is a bookmark section next. If you bookmark a menu item in the Common or Advanced Settings section of the Admin menu, those bookmarked features will appear here. You can always remove them by clicking the icon next to the feature name, which will keep the feature but remove it as a bookmark.

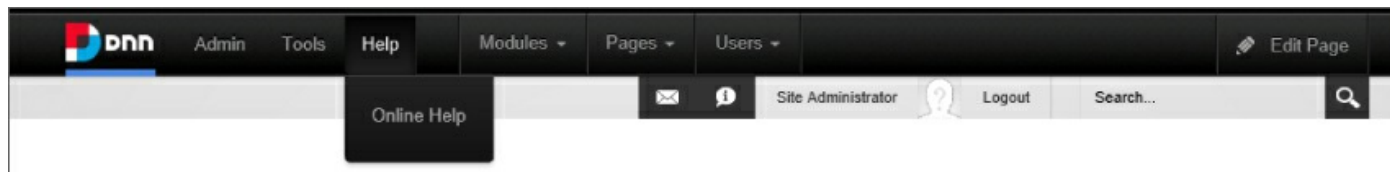
Tools

The Tools menu will look a bit bare to you as a site administrator. This menu is mostly useful only to a host or superuser account, where it displays more options. However, if you want a shortcut way to quickly upload a file, you can do so from here with a single click.

While this might appear to be convenient, it is much more common to upload files from other features and views in DNN.

Help

If you need help while logged in as an administrator, the Help menu is not a bad place to start (see [Figure 4.3](#)). It allows you to access the Online Help and Getting Started view. Online Help can also be accessed from the Community  Learn section on the DNNSoftware.com website. The Getting Started view aggregates some helpful information and links to help you learn the basics about DNN so that you can do some of the most common tasks. It also has shortcut links to places like the Community Exchange and Community Voice sections on DNNSoftware.com. These are areas where you can get a question-and-answer form of help and submit enhancement ideas to DNN Corp.



[Figure 4.3](#)

Modules and Pages

The Modules and Pages menus are the same ones that you see as a user with page-level editing permissions. Refer to [Chapter 3](#) for more information about these menus.

Users

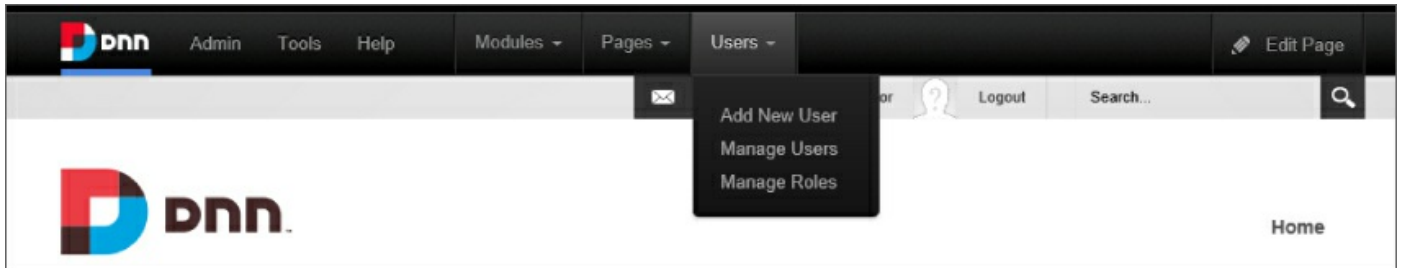
One of your primary responsibilities as the site administrator will likely be to manage users on your site. You might need to create, edit, or delete users. You might also be required to add and remove abilities from users on your site. Everything that you need to do in these examples can be done from this one menu.

We'll discuss the features of the menu options in great detail later in this chapter, but the options you have in the Users menu include the following (see [Figure 4.4](#)):

- **Add New User:** Add New User is exactly what it sounds like. If you click this option, a pop-up window will appear where you can create a new user account. If pop-ups are disabled for your site, this same view will open,

but in a new page.

- **Manage Users:** You will want to use the Manage Users feature if you want to search for, edit, delete, or otherwise manage any user account on your site.
- **Manage Roles:** When you manage roles, you can of course create, edit, and delete roles. However, using either Manage Roles or Manage Users, you can also assign and revoke security roles from user accounts.



[Figure 4.4](#)

Edit Page

As with Modules and Pages, the Edit Page menu in the Control Panel is no different for administrators. Refer to [Chapter 3](#) for more information about the options.

All other portions of the content-editing experience will be consistent between administrators and content editors of the site. The only time there are differences is when the content editor has a limited number of capabilities assigned through permissions, which should be something you enforce as a best practice anyway.

Admin Menu Features

The Admin menu is loaded with features that are meant to make your job as a site administrator fast and easy, starting from the moment you are given the ability to log in to your DNN site. Everything that you need is literally one or two mouse clicks away. In this section, we discuss each of the most common and useful features in the Admin menu.

Common Settings

It was mentioned previously, but the first section of the Admin menu consists of views and features that are categorized as Common Settings. These are the settings and site management tools that you need to access on a regular basis. These features are only a click away since they're in this section.

Event Viewer

The Event Viewer (see [Figure 4.5](#)) allows you to know what's happening on your website at all times. You may see this feature sometimes referred to as *Event Log* in older documentation and community resources. There have been some releases where it was indeed called Event Log in the Admin menu, the API, and the database. Also, the same name is used for the event viewing utility in Microsoft Windows. If you see or hear either term and the person is referring to DNN, it's likely that they're talking about the Event Viewer.

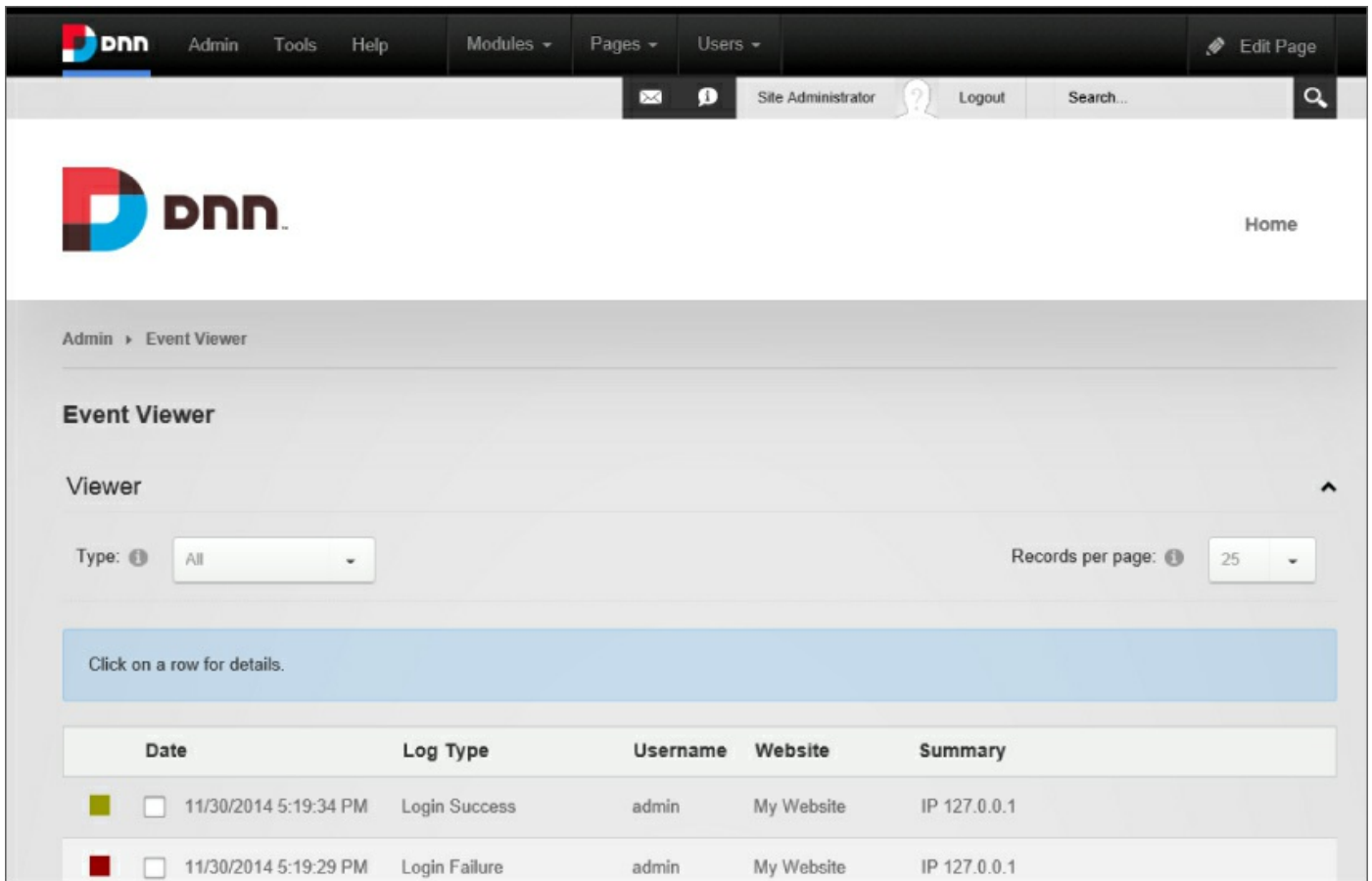


Figure 4.5

There are a lot of events that are relevant to site administrators that will be stored in the Event Viewer for you to see at any given time. If you're logged in as a host/superuser account, you can see the events for all sites on the same installation of DNN (should they exist). However, site administrators will see the events only for their own site that they manage, and even then, the messages you see will be filtered to show you messages that are relevant only to the site administrator.

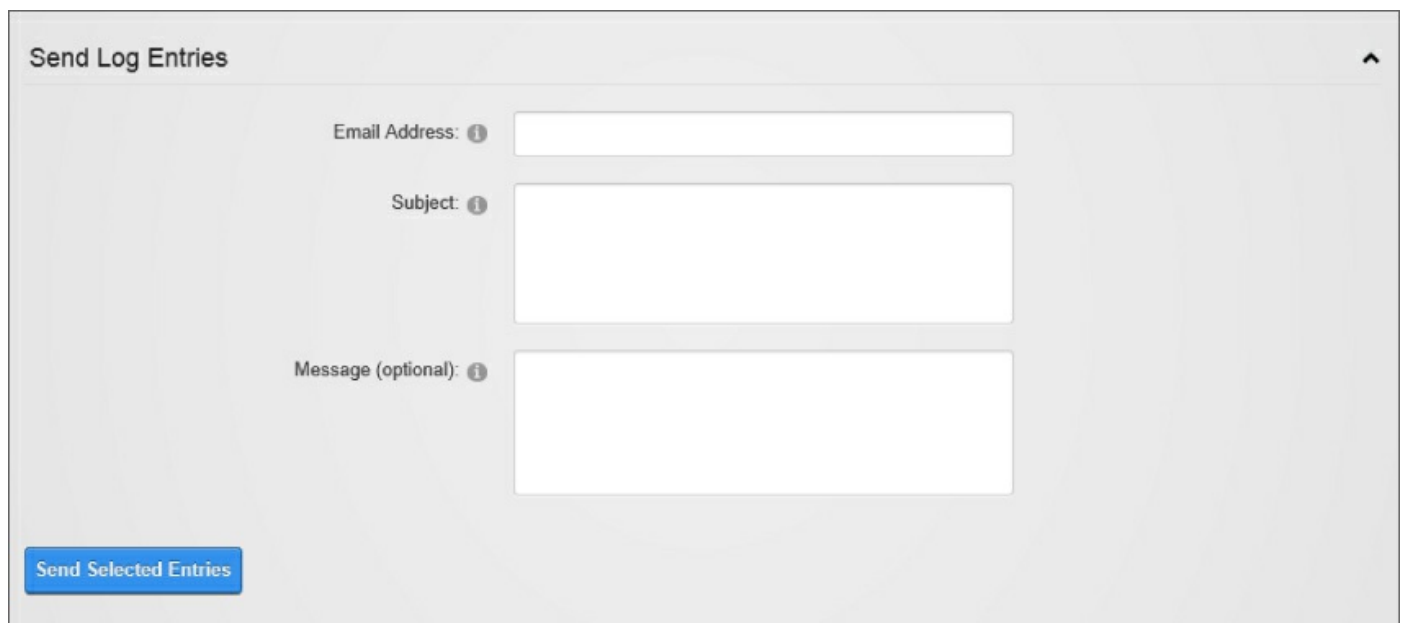
On the surface, the Event Viewer may seem very simple, and it is. You have a listing of the most recent events immediately visible on page load. This list is going to be any number of pages with 25 events on each page of results. A paging feature at the bottom of the list allows you to move from page to page.

If you want to filter the events to see more or fewer events, you have two controls at the top of the listing to help you. First, you can filter by the type of event that you want to see. For example, you might be interested only in seeing instances of site users attempting to log in and failing. To see this, simply change the Type drop-down list to Login Failure, and only those events will be shown to you.

In other cases, you might want to see a larger list of events. There are many reasons for this. One reason might be searching for something specific using built-in features of your web browser, such as using Ctrl+F to “find” something in the page. As an example, you could switch the Event Viewer to show 100 or 250 events by changing the Records Per Page drop-down list. The page will reload with your preference chosen and applied.

The Event Viewer also has a legend at the bottom that explains the colors that are applied to various events.

The final feature that is on the surface of this module is the Send Log Entries section (see [Figure 4.6](#)). You’ll find it at the bottom of the page, and it will be collapsed by default. When you expand it, you can select any number of events in the listing and then send them all at once to the specified email address. It also allows you to send a message with the events you send. This feature is useful for sending errors to a support resource to inform them to get more details on how to respond to events that you may be concerned about.



The screenshot shows a web interface titled "Send Log Entries" with a close button in the top right corner. Below the title are three input fields, each with an information icon to its left: "Email Address", "Subject", and "Message (optional)". At the bottom left of the form is a blue button labeled "Send Selected Entries".

[Figure 4.6](#)

Additional Event Viewer Actions (Host/Superuser Only)

There are a handful of additional features that the Event Viewer has that you might not know about or even discover unless someone told you that they existed. These additional features allow you to delete any selected events from the listing, clear the entire list of events, and edit the log (viewer)

settings (see [Figure 4.7](#)).

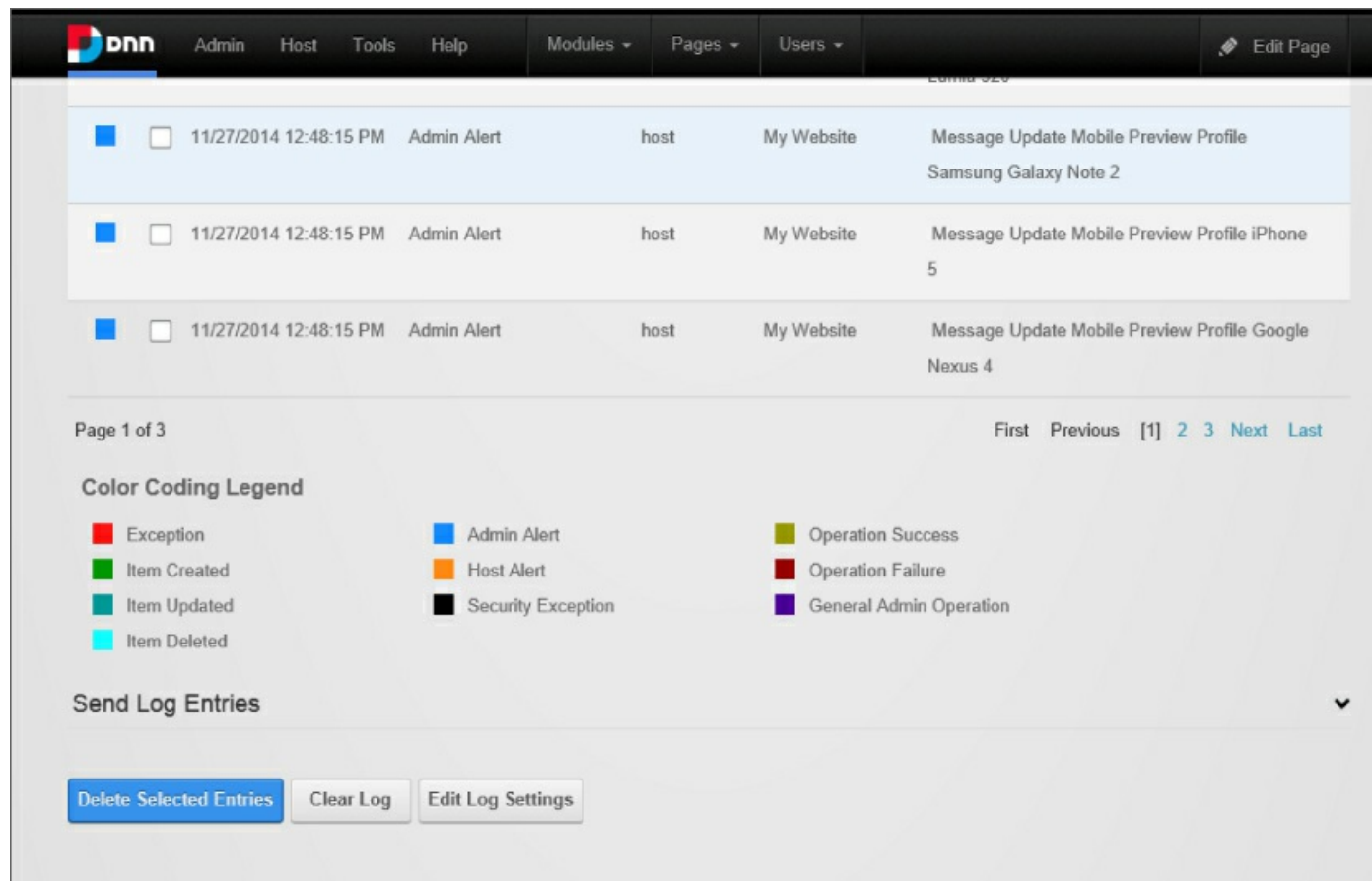


Figure 4.7

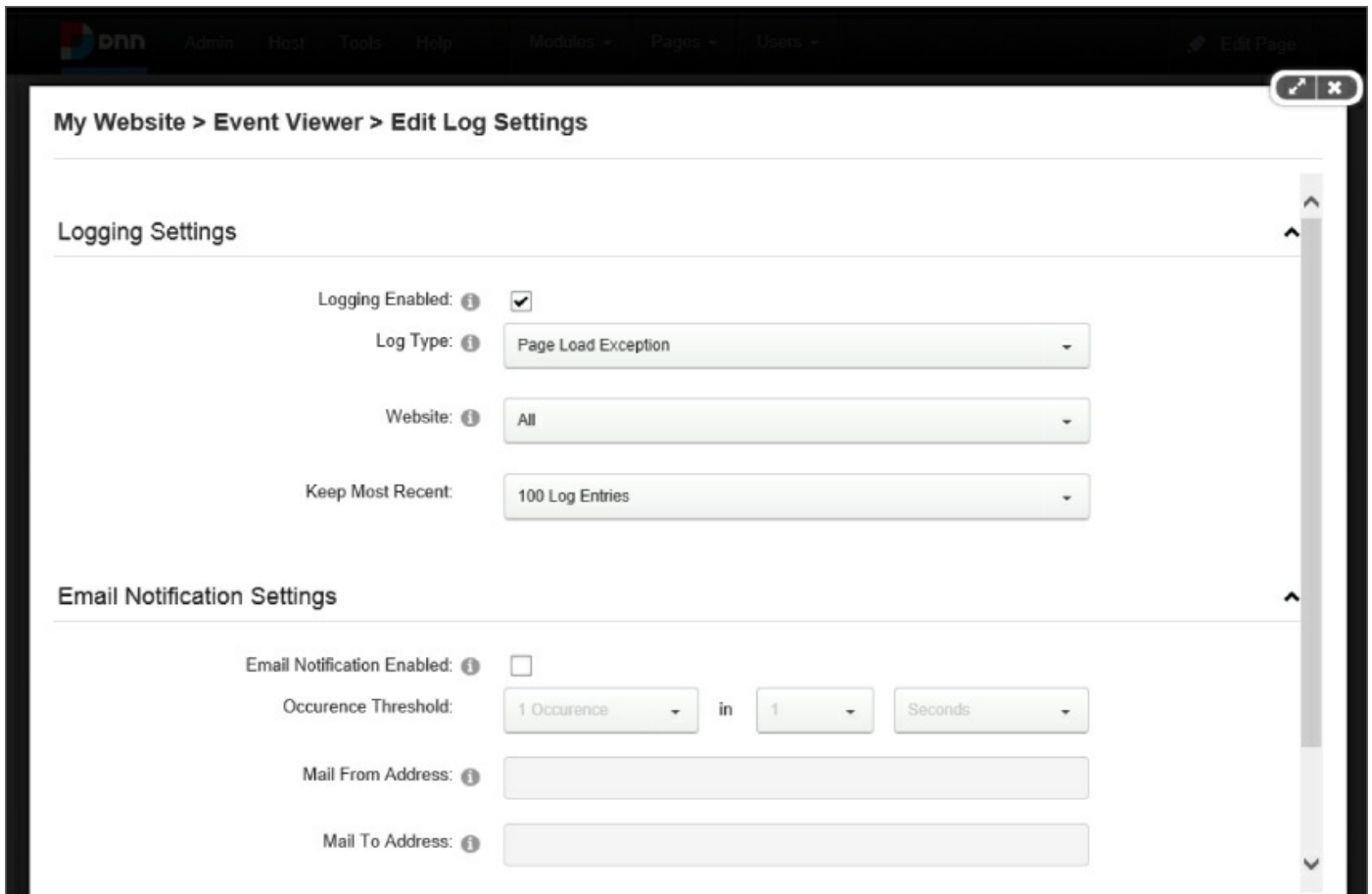
While you can delete individual events from the Event Viewer, there generally isn't an actual reason to do this on a production website. However, if you're developing DNN extensions, this feature could be useful for removing duplicate errors or other events to make it easier during your development process.

The Clear Log button is a useful feature if you don't already have a process in place to clear the events from your system. This is important because the Event Viewer can fill up with events quite quickly. The more events you have logged, the slower your site will be. For example, any average site will likely see a noticeable delay in pages loading when there are 100,000 or more events saved. The Clear Log button is a quick way to remedy any performance issues arising from this issue, but it is not a best practice in most environments. A more business- and process-friendly approach to keeping the Event Viewer clean and to keep the number of logged events to a minimum in the production database is to have a regular process created and running in your database, use a different logging provider, or use a DNN scheduler item.

In some cases, your organization might create a routine in SQL Server to move events to another database for archiving.

Edit Log Settings is a useful tool that allows you to edit any of the individual types of events that can be logged. You can enable or disable them to prevent or allow them to be logged and subsequently appear in the Event Viewer, limit the sites that can log the specific event, determine how many to save, and send the events to an email address when they occur. Most of these options should be self-explanatory. As an example, the default settings will create an event when a folder is moved. However, you won't see an event entry when a folder is deleted. If you want to see when this happens, you now know where you can enable this.

The most compelling feature in this view is the Email Notification Settings section, as shown in [Figure 4.8](#). This allows you to choose to subscribe an email address to the event. Whenever the event occurs, it will be sent to the specified email address. This is useful for many functions. Imagine your support team automatically getting an email when errors begin to occur or your security team getting notified upon unsuccessful login attempts. Or, maybe you want to know whenever another user updates the site settings or deploys a module to a page. These things are all possible, but again this feature is available only to host/superuser accounts.



[Figure 4.8](#)

File Management

If you have been using DNN for any length of time, you may have noticed that File Management recently received a major makeover. This wasn't just a makeover, it was a full on re-write and re-imagining of the feature. The module that was formerly on this page was called the File Manager module. The new module is named the Digital Asset Manager (DAM) module, and it boasts a ton of new features that will definitely make document management easier.

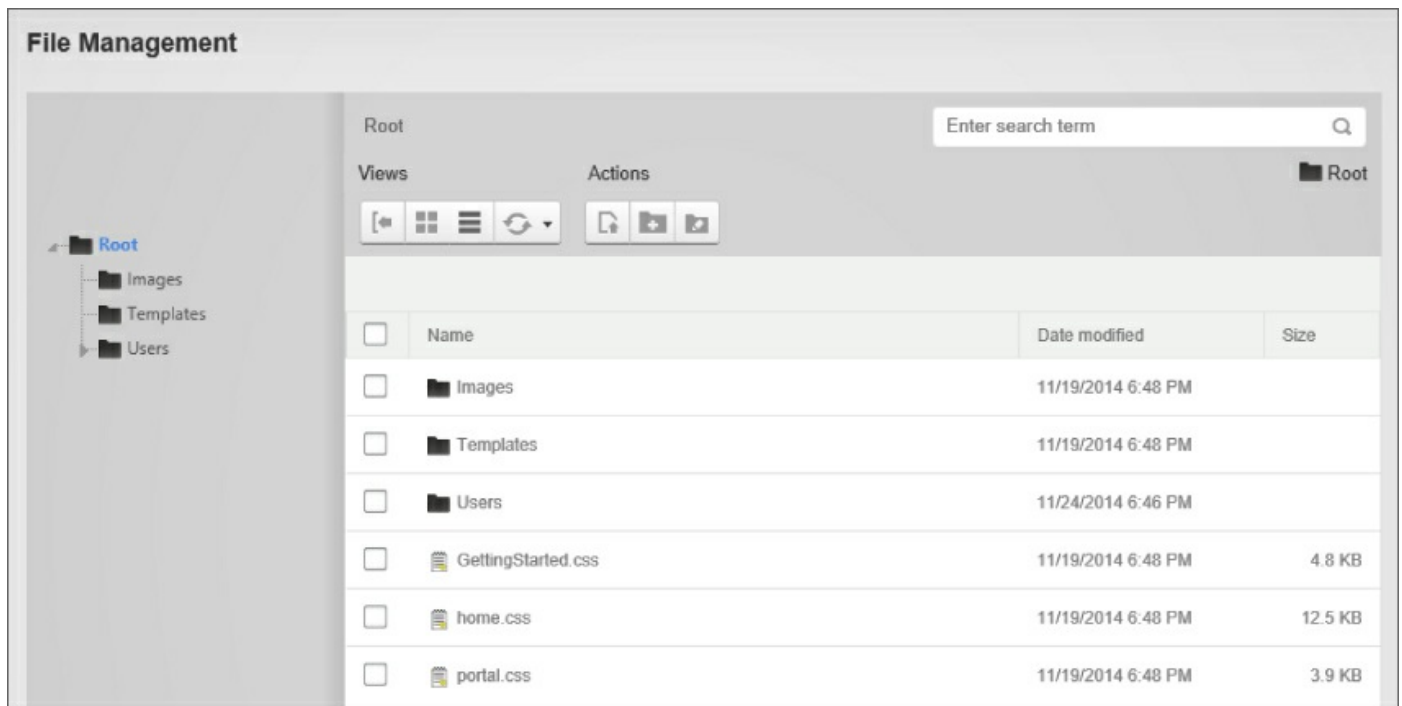


NOTE

This module has a commercial counterpart in the commercial versions of DNN known as Evoq Content and Evoq Content Enterprise. See [Chapter 20](#) for more information on the differences of this feature.

Nearly everything you could need to do with files and folders in DNN can be done from this view. You have the ability to upload, edit, copy, move, and delete files and folders. You can also control where files are stored and what permissions should be applied to which folders.

The DAM module has its own Control Panel of sorts in that there are a few features in the header area of the module that are intended to help you manage your files and folders. When no files are selected, it will appear as shown in [Figure 4.9](#). When one file is selected, a handful of buttons will appear to help you manage that file, as shown in [Figure 4.10](#). Finally, if you select more than one file, [Figure 4.11](#) shows you the subset of those same file management buttons that remain.



[Figure 4.9](#)

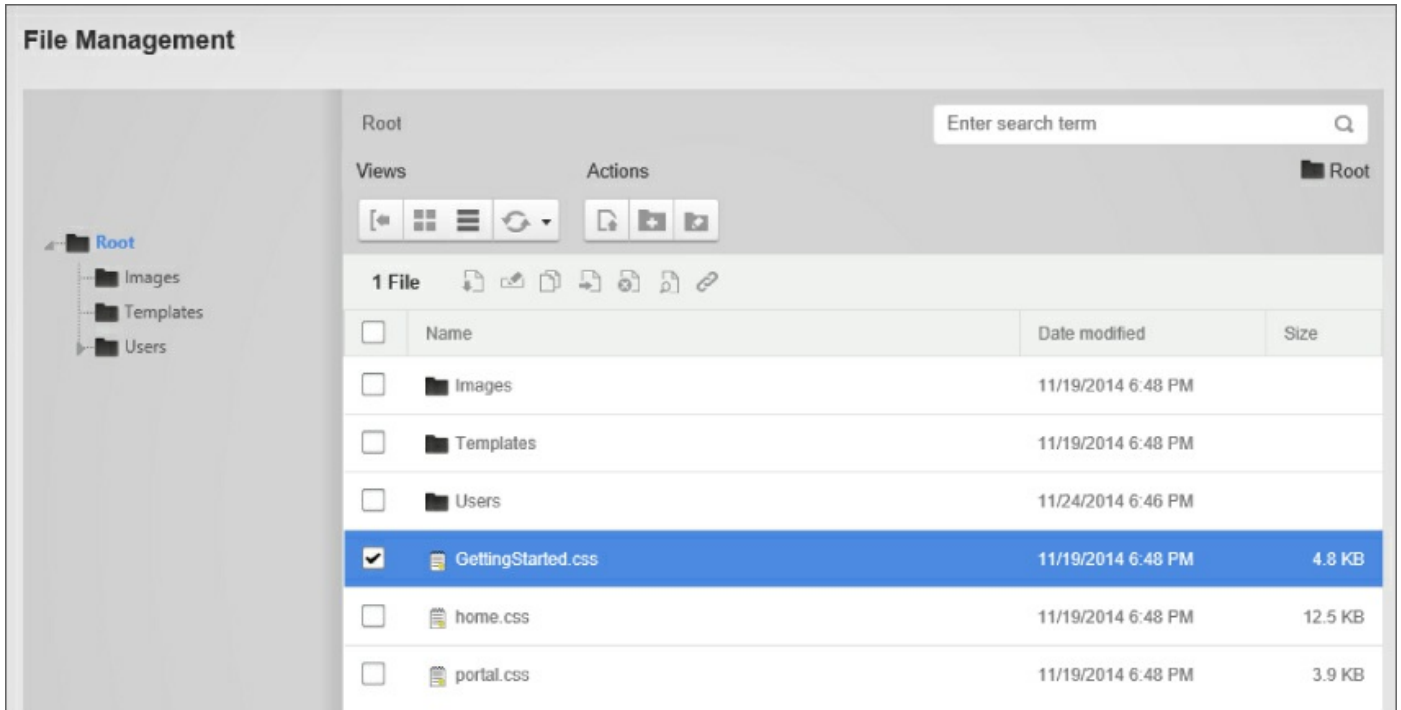


Figure 4.10

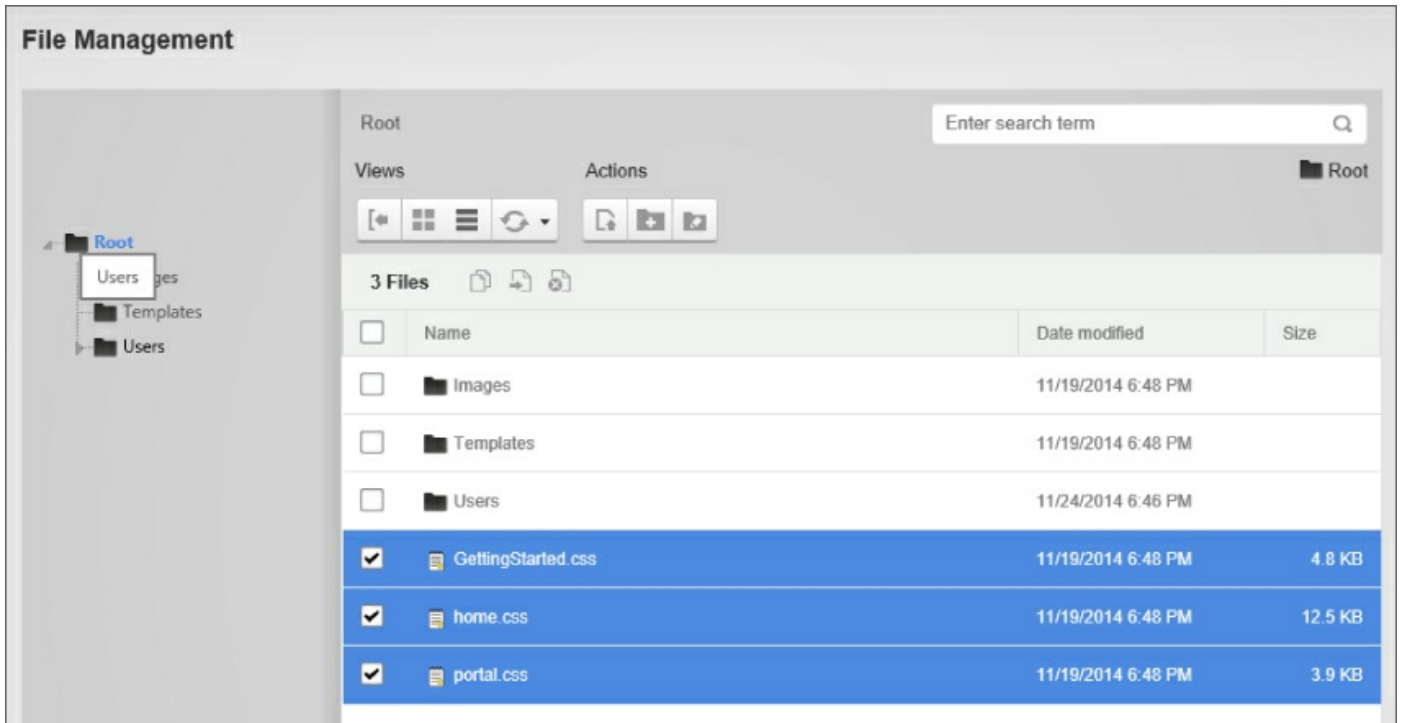


Figure 4.11

Table 4.1 explains what each button allows you to do.

Table 4.1 Digital Asset Manager Control Panel

Number	Name	Description

1	Search	The search allows you to filter the listed results when you're looking for specific files in a folder that has too many to glance through easily.
2	Toggle	Clicking this button will hide or show the folder panel that's on the left side of the DAM module, depending on whether it was already hidden.
3	Icons	This button will display the listing of files in an icon view. This is useful if you prefer to look at your files in a "Large Icon" style of view, similar to Windows. If any files are selected prior to clicking this button, the selection will remain.
4	List	The list view is the alternative view to the icons view. This is the default way that files are displayed and shows more files and information at a glance. Also, in Evoq solutions, this view may display information such as subscriptions or workflow.
5	Refresh Folder/Sync	This button is actually a menu. When you click it, you'll be given three options. Refresh and Sync this Folder will simply scan the selected folder and update the DNN database with any file changes found on the server. If you choose Sync this Folder & Subfolders, all the subfolders of the selected folder will be synchronized as well.
6	Create New Folder	Clicking this button will open a dialog box that will allow you to specify the details for the new folder you want to create. This option is discussed in more detail later in this chapter.
7	Upload Files	When you choose to upload files, this button will open a pop-up window where you can either drag and drop the files you want to upload or select the files the traditional way by browsing the file system on your computer.

Context Menus

You don't have to do it for most actions, but whenever you want to manage a file or folder, it's usually easiest to simply right-click the file or folder in

question. This will result in a menu appearing that's usually referred to as a context menu, like shown in [Figure 4.12](#).

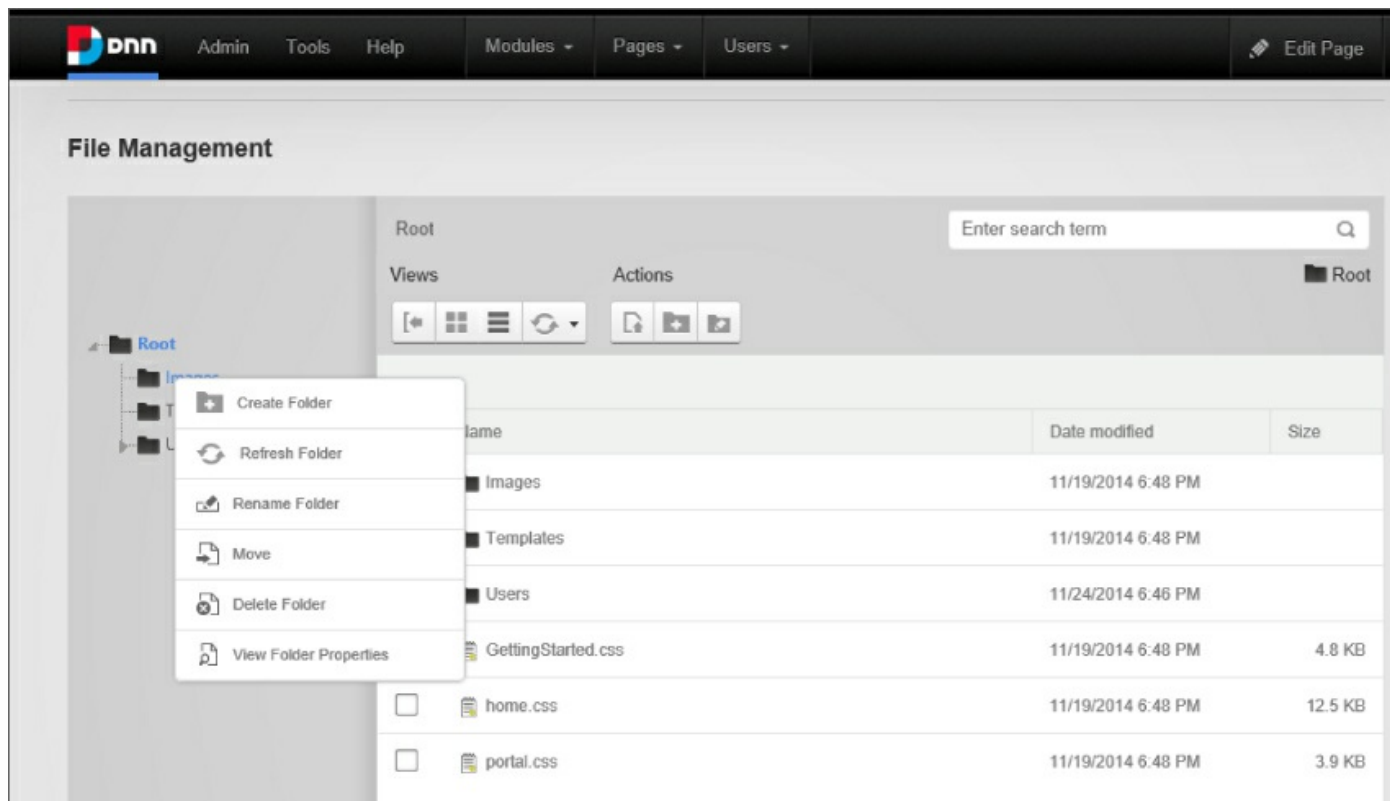


Figure 4.12

If you right-click any folder other than the Root folder, you will get a menu that allows you to do the tasks listed in [Table 4.2](#).

Table 4.2 Folder Menu Options

Menu Option	Description
Create Folder	This allows you to create a new folder that will branch off of the folder that you chose to right-click.
Refresh Folder	DNN will look in the folder on the file system to update the view with any changes that might have been made outside of DNN.
Rename Folder	You can give the folder a new name. Be sure to do this as little as possible, as it will likely result in broken links and images.
Move	This allows you to move the folder to another location in the file structure. As with renaming, you should be aware that this action might result in broken file paths.

Delete Folder	Deleting the folder should be self-explanatory. If you choose to do this, you again should be aware of potential broken links. This is a permanent action.
View Folder Properties	A pop-up will appear where you can edit the various options for the folder. In DNN Platform, that includes renaming the folder and modifying the permissions for the folder.

File Synchronization

Files are synchronized automatically over time with the default configuration of DNN. However, you can use the DAM module Refresh Folder feature to make this happen immediately. This is a useful feature if you're uploading files by any means outside of DNN, including Windows XCOPY, file server operations, WebDAV, and FTP. If the synchronization hasn't happened either manually or by way of the scheduled job in the background, any file or folder changes made outside of DNN will not be reflected in the views that display files and folders to content editors.



NOTE

XCOPY is a technical term to describe the process that uses XCOPY or similar operations to automatically move files from one location to another by Windows or a Windows application. FTP is a common way to transfer files from a computer to a server at a remote location.

The synchronization and display/selection of files depend on your host settings, discussed in [Chapter 5](#). If a file type is not specified in the Allowable File Extensions setting, then the file type will never be shown in the DAM module or in any other view in DNN. This feature is often referred to as the file's “white list.”

Uploading Files

Using any option to upload files to the DAM module will display the same upload window (see [Figure 4.13](#)). This window allows you to drag and drop multiple files to upload them to DNN in a single action. You can also upload files one-by-one by clicking the Choose Files button.

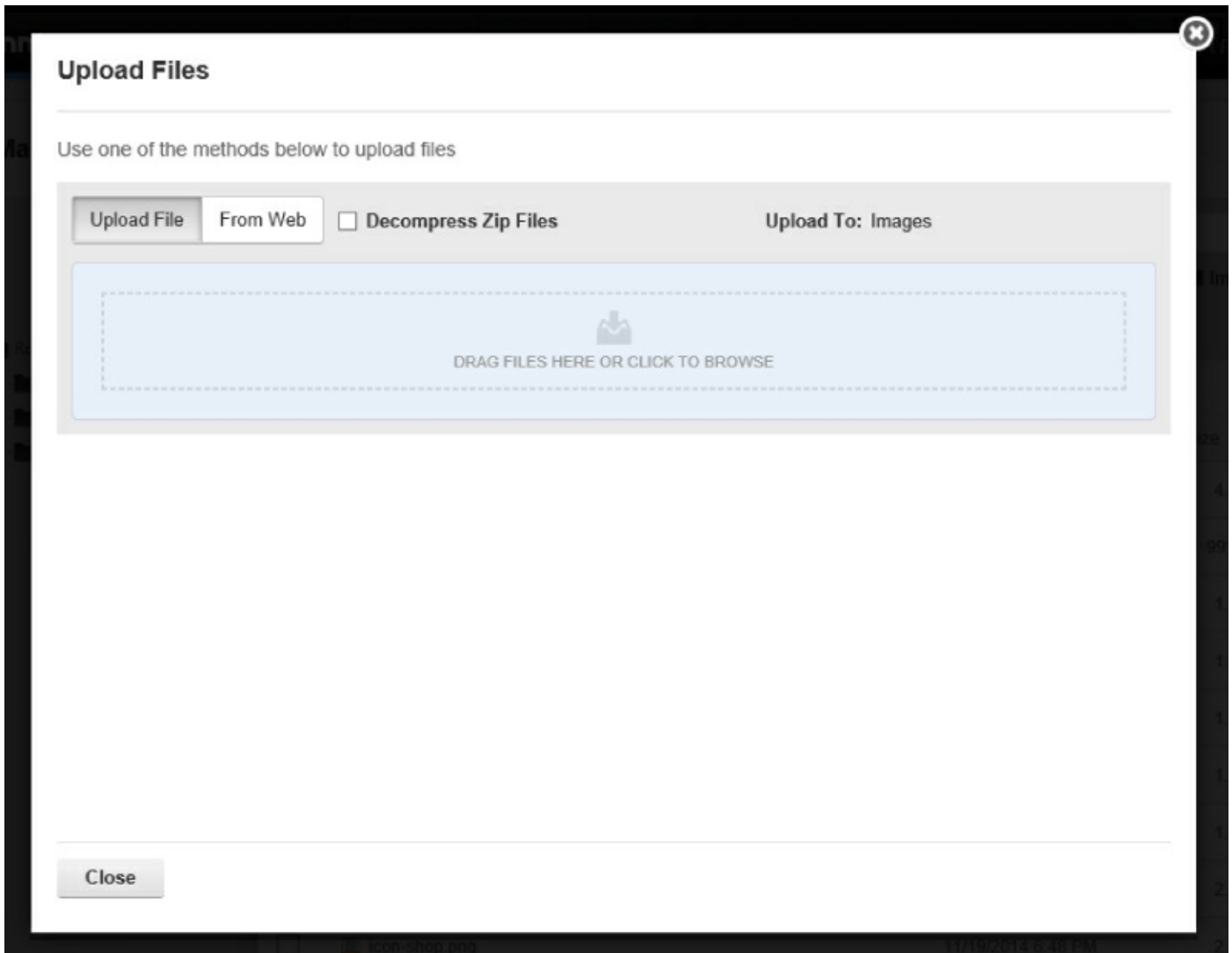


Figure 4.13

If you choose to upload any archive files, there is a checkbox that asks you how you want the upload to occur. Archive files are also known as ZIP files and the filename will end with .zip.

DNN has long supported something called the “bulk upload” by adding multiple files to an archive file. If you select Decompress Zip Files, the ZIP file will be uploaded, but it will also unzip it and put all of the contents of the ZIP file into the same folder. Otherwise, the ZIP file will be uploaded like any other file, as shown in [Figure 4.14](#).

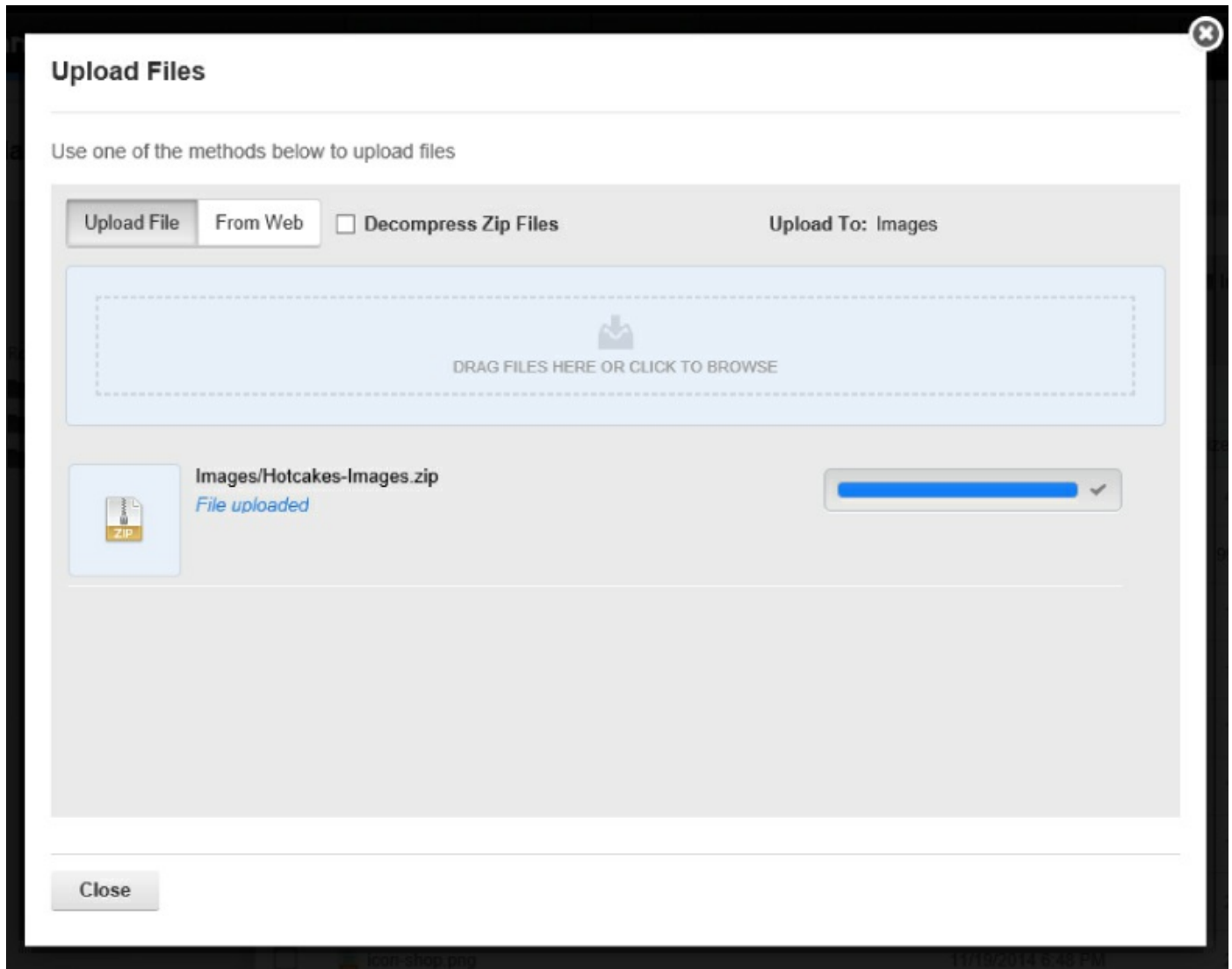


Figure 4.14

Creating Folders

When creating a folder, you see three fields—Parent Folder, Folder Name, and Folder Type. The latter two options are configurable. The Parent Folder is decided for you based upon the folder you chose to create the new folder under. The name of the folder should be self-explanatory. Give your folder whatever name you like, but be sure to name it logically so that the content editors can easily tell what it's for.

The other option you can edit when creating a folder is Folder Type. You have three options for this field out of the box, and they include Standard, Secure, and Database. They are defined for you in [Table 4.3](#). That being said, this is one of numerous extension points.

Table 4.3 DNN Platform Folder Providers

Folder Provider	Description
Standard	This folder provider works the same way that any typical folder works for any website. The path to files in this folder is easily read by humans, and access to the files is very straightforward. You can apply permissions to this folder, but there are certain use cases where direct access to the files can be guessed and therefore bypass any permissions you assign. This is the most common folder provider you would want to use in most instances.
Secure	This folder will continue to store files in the DNN file system just like the Standard folder provider. However, this folder will rename the files in the file system itself, which will obscure the location of the file from people and apps that might be reading the links generated for the file. This new filename is seen and used only by DNN. People managing the files in DNN will continue to see the original filename. Also, these files will be accessible only using an obscured URL and not a human- or SEO-friendly URL. This folder type allows you to ensure that any permissions assigned to this folder are used in all use cases.
Database	The Database folder provider works in a similar manner as the Secure folder provider with a specific exception. The files that you save and access will be stored in your database. While this is the most secure option available, it has trade-offs. File access will be slower and the size of your database will grow quite quickly. If either of these things is an issue, it may be more logical to use the Secure folder provider.

If you want another way or place to store your DNN files, you can upgrade to an Evoq solution or find a third-party option in the DNN Store or Community Forge. There are many types of folder providers out there, with more being created regularly. The folder provider might allow you to save files in a shared network folder or in one of many cloud storage services (such as Box, Dropbox, and so on). Regardless of the type of folder you choose to use, you'll work with the new option the same way. The only difference is where or how the file is stored, but you will generally not know any different.

WARNING

Once you create a folder using a specific folder provider, you cannot switch to another type of provider. The folder must be deleted and re-created using the intended folder provider.

Folder Permissions

You can view and change the permissions of any folder (provided that you have the permission to do so) by right-clicking the folder and choosing View Folder Properties from the context menu. The permissions grid will be on the second tab, labeled Permissions (see [Figure 4.15](#)). This permissions grid works the same way as all other permissions grids in DNN.

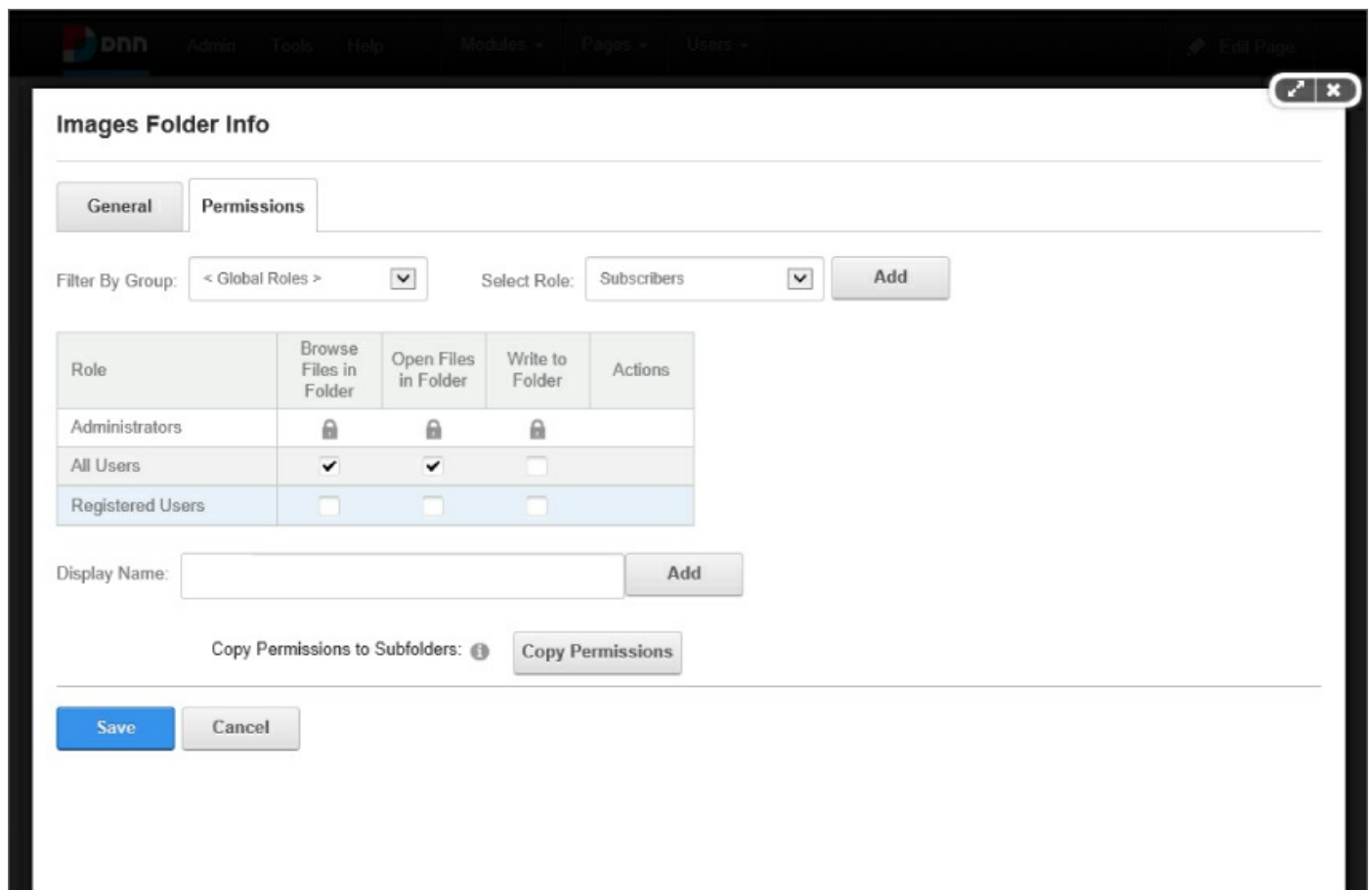


Figure 4.15

Folder permissions allow you to determine who can see folders and files and who is allowed to modify the contents of a folder. In DNN Platform, the permissions are not granular, so you have three options when modifying the

permissions of any folder. Also, permissions can be assigned only at the folder level, and not for individual files. [Table 4.4](#) explains what each column will allow a user or group of users to do, if allowed.

Table 4.4 Folder Permissions

Permission	Description
Open Files in Folder	This allows the specified user(s) to view the files when they are displayed on the site in any way. When you first install DNN, “All Users” will be able to do this from the Root folder by default. This also allows users to access the files directly if they know the URL to the file.
Browse Files in Folder	Users included in this permission are allowed to browse through the folder to select files in the DNN views for things like creating links in HTML and other features. Each user is provided with his own personal folder upon account creation. This user folder will automatically have this permission applied to that folder for him.
Write to Folder	When users are granted this permission, they are able to perform all file and folder management tasks on the respective folder. Think of this as a “full control” kind of permission. The folder can be renamed, moved, and copied; subfolders can be created; files can be managed; and so on.

Evoq Version of the DAM Module

While the DAM module found in DNN Platform should work for the majority of sites, there is a more advanced version of this module found in Evoq Content and Evoq Content Enterprise. In addition to what we've already discussed, the commercial version of the DAM module includes

- Granular folder permissions
- Workflow to approve files
- Versioning of files
- Tagging of files
- Expiration of files
- File and folder subscriptions

- Additional folder providers (Windows Azure, Amazon S3, and UNC file share)

Page Management

Pretty much every website starts out as one thing first and foremost: a collection of pages that constitute a web presence. Each page will have a purpose (we hope) and serve it well. As you have seen after the default installation of DNN, you are even given a home page to work with on the first load of your new site. Your next step might be to create additional pages, such as About Us, Contact Us, and more.

As your site continues to become successful and grows to meet the needs of your customers and other visitors, you will undoubtedly begin to have more and more pages on your site. You'll have so many that at some point you will need to manage them from a centralized location. In other cases, you might have pages that are hidden from the menu and are not easily accessible. This is where the Page Management feature comes in (see [Figure 4.16](#)).

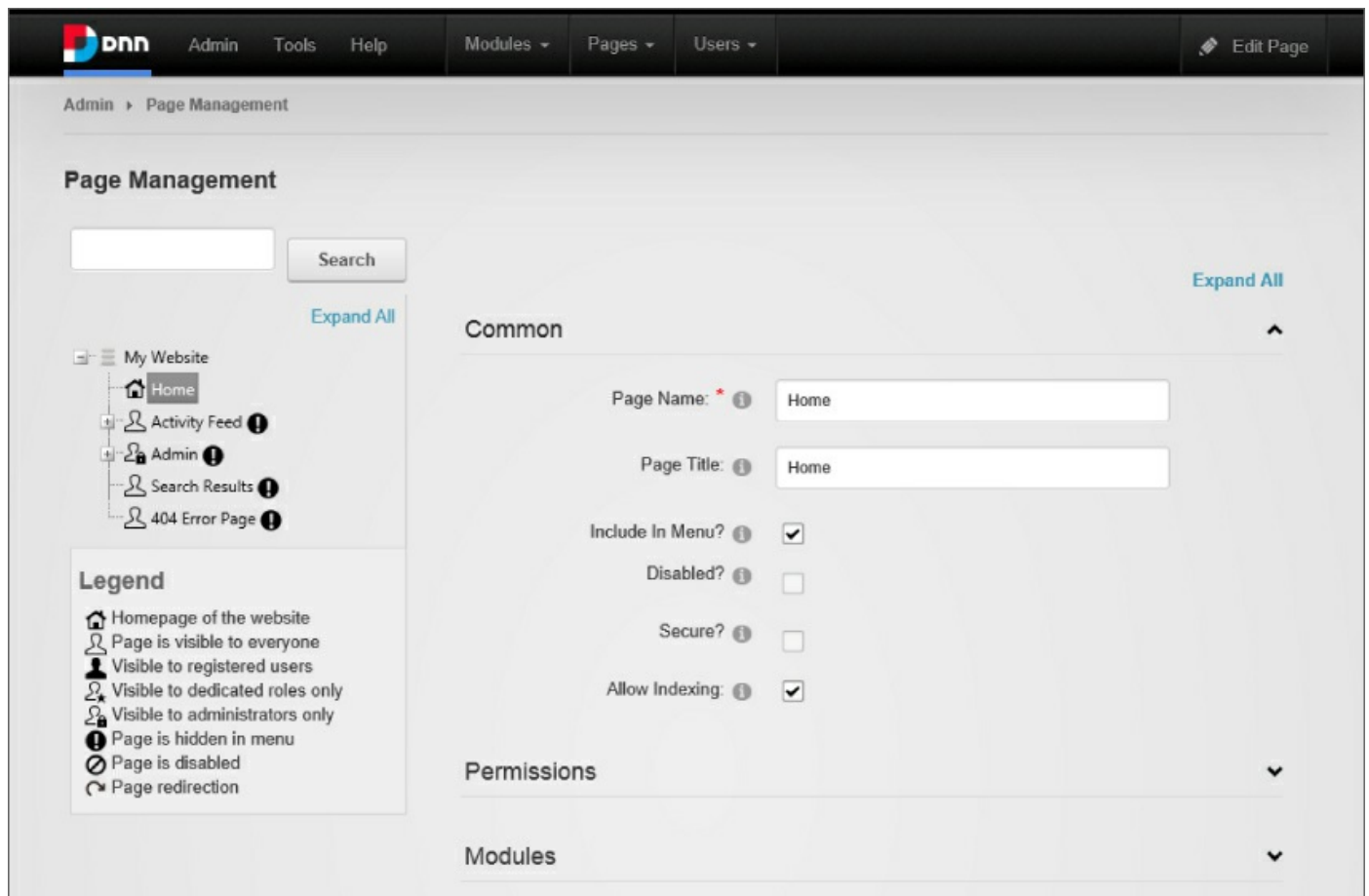
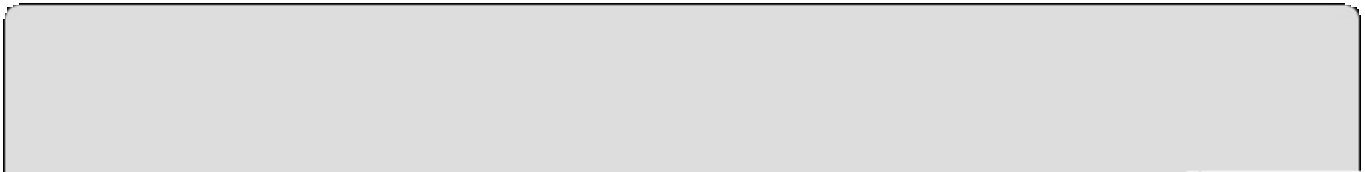


Figure 4.16

The Pages module is found in the Admin menu on the Page Management page. It includes three areas for easy management of your pages.

If you're not already using the Page Management module regularly, you should consider doing so. This module makes common and repetitive tasks painless. It allows you to manage any page on the site from a single place, without having to navigate to each individual page, thus saving you time. That being said, page management from the page level is not dead. There are still plenty of reasons and use cases where it will remain necessary to navigate to an individual page to manage its settings, or you might already be there and navigating to the Page Management page would be an extra step.



TIP

This is a great module to consider opening up for non-Admins who have a global scope of content-management tasks. An example of this is an SEO expert or consultant. This kind of content editor will likely spend a lot of time in this module.

Navigation

The Navigation area of the Pages module allows you to search for and choose pages to manage. This is presented to you in a format known as a “tree view.” It allows you to drag and drop pages and expand or collapse certain sections of the site. Expanded areas will be visually indicated by a minus icon and the child pages being indented from the parent page. When a section is collapsed, the minus icon will switch to a plus icon and the child pages will be hidden from the view.

You can choose pages that will load the page settings for the chosen page on the right pane. You can drag and drop pages to move where they appear in the site hierarchy. Finally, you can also right-click any page to perform quick administrative tasks without having to waste time looking through settings or views for the same feature.



NOTE

As of DNN 7.1, you can safely drag and drop pages in the site hierarchy without worrying about the URL changing and ruining your SEO work. The original URL will be retained and properly redirected if inbound links are still using it.

Legend

The legend shows you some of the iconography with a description of what the various icons mean. This area has no additional features, but it is helpful to understand why your pages might have different icons next to them.

Page Settings Pane

The Page Settings pane covers most of the right side of the Pages module and will simply be “whitespace” until you select a page from the navigation area. Once you select a page, this area will be loaded with the settings of the chosen page.

The Page Settings pane is reorganized in a contextual way within the Pages module to make settings easier to find for those folks who just manage pages.



NOTE

When using this module, you will notice that some features are missing from the Page Management module that appear in the actual Page Settings view. This includes the Copy Permissions to Descendants setting, URL management, and more. In these cases, you might want to right-click the page to see the Page Settings pop-up or navigate to the page and edit its settings there.

Context Menu

A context menu appears when you right-click a page in the navigation area of the module. This context menu was designed to save you time. For example, creating your home page might not have been intuitive. This context menu now makes that task a simple click.

There are a number of actions that are available in this menu, depending on the selected page. As an example, you no longer see the Make Homepage option if you right-click the home page itself—it wouldn't make any sense. Another example of this is the Hide in Navigation option. When the page is already hidden, the context menu will switch that menu option to Show in Navigation. [Table 4.5](#) provides more insight into each option.

NOTE

Many of the actions that you take on pages in the Pages module use a technology called AJAX. This allows you to save your changes quickly and not have to worry about the page reloading. However, this also will require you to navigate to another page or manually refresh the page in some cases to see your changes take effect. The primary example of this is when you add, delete, or move pages in the navigation.

Table 4.5 Pages Module Context Menu Options

Name	Description
View Page	This is shown on all pages and allows you to navigate directly to the chosen page in the same window.
Page Settings	Like View Page, all pages get this option. Choosing this option will open the settings for the chosen page in a pop-up window instead of loading it in the Page Settings area of the Pages module. This is essentially the same view you see if you navigate to the page directly to edit its settings.
Delete Page	This option is shown to all pages except for Admin and the page that's selected as the site home page. When selected, DNN will attempt to delete the page, but not before prompting you with a confirmation window.
Add Page(s)	You can add pages from any page, including the top-level website node. This feature is discussed in detail later in this chapter.
Hide from Navigation	When a page is shown in navigation, you will see this menu option. Selecting this option is the same as checking the checkbox for the Include in Menu setting in Page Settings. When this option is chosen, the page will be hidden from the site navigation, but any users who are allowed to view the page will still be able to see the page.
Show in Navigation	This menu option appears if the page is hidden from the menu. It is the opposite of the Hide from Navigation option. When selected, the chosen page will appear in the site navigation. Again, this feature is directly linked to the Include in Menu setting of Page Settings.

Disable Link in Navigation	If you want to disable a page from being viewed but still show it in the navigation, select this option. This is useful when you want to use a page to anchor other pages for a drop-down list effect in your navigation. Only administrators can view this page if chosen. This feature is directly linked to the Disabled setting in Page Settings.
Enable Link in Navigation	This menu option is the opposite of the previous setting and will be shown only if the page is already “disabled” from the site navigation.
Make Homepage	Great care and consideration should be taken before selecting this option. Changing a page to be the home page has effects that could spread across more areas than you realize. Changing the home page will affect where users end up when certain pages are not found and when clicking the site's logo. Generally, this setting is used only once during the lifetime of a website.

Evoq Version of the Page Management Module

The Pages module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Recycle Bin

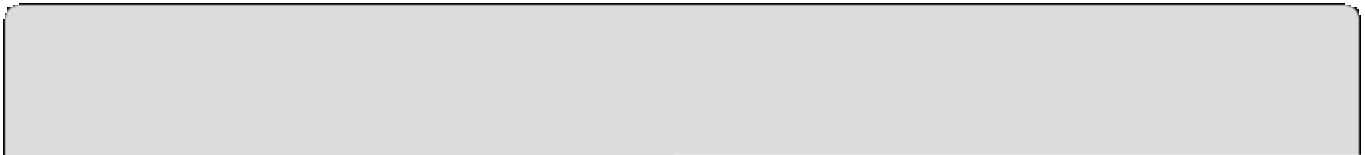
The Recycle Bin is a very useful and often overlooked feature of DNN, especially for new users. Oftentimes, a newly designated administrator for a site will simply assume that when something is deleted, it is gone. The next logical step would probably be to re-create the content that was just deleted. This could lead to frustrations with DNN. The Recycle Bin allows you to restore the content in most cases, which can save you time and frustration.

As of the writing of this book, the Recycle Bin will seem quite simplistic. There are suggested updates to this module in Community Voice that will make it look far better and offer more capabilities than what is currently available.

TIP

The updated version of the Recycle Bin is likely to be a community contribution and originated as an idea in Community Voice. Anyone can add their own ideas for enhancements to DNN. Simply go to www.DNNSoftware.com/Voice to submit your idea.

The Recycle Bin (shown in [Figure 4.17](#)) has two types of objects that can be restored—Pages and Modules. When you delete a page or a module, it will show up in the respective list. You can switch which list you're looking at by clicking the tabs. Clicking one of the pages or modules will allow you to restore the selected item(s) to their original place in your site.



NOTE

You cannot restore files or users from this view. Restoration of users will be discussed later in this chapter, and restoring deleted files is not supported at this time. However, files can be restored from a site backup if you have a backup plan in place. You should check with your system administrator and/or web host to ensure that you do have a backup plan.

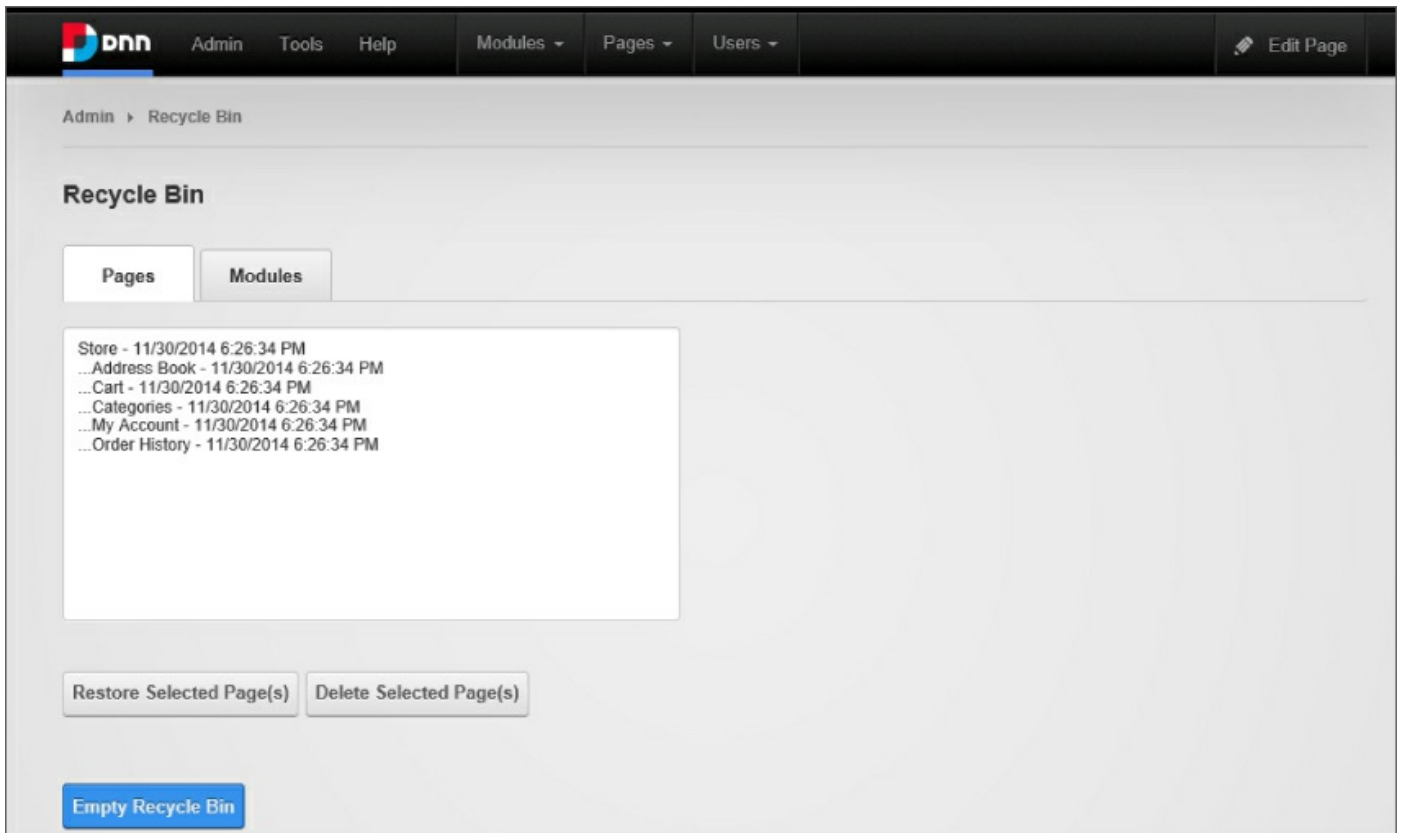


Figure 4.17

Restoring Pages and Modules

When you're on the Pages tab, the two buttons below the selected list will be labeled Restore Selected Page(s) and Delete Selected Page(s). When the Modules tab is selected, these buttons will instead delete or restore the select module(s). This is dependent on there being pages or modules selected to begin with. If no pages are selected and you click the Restore or Delete button, nothing will happen. This happens on the Modules tab as well.

When you choose to restore a page or module, DNN restores it to the original place. In the case of pages, the page will be restored to the same place in the site hierarchy. In the case of modules, the module will be restored to the page from which it was deleted. Any content or settings associated with the module or page will also be restored.

If you want to restore a module that was on a page that was also deleted, you need to first restore the page. If you do not, DNN will prompt you to do so. Once you restore the deleted page, you can then restore the modules.

It will be fairly easy to recognize the page that a module is associated with by looking at its description in the listing, as shown in [Figure 4.18](#). The page name precedes the module name, followed by the date that the module settings were last modified.

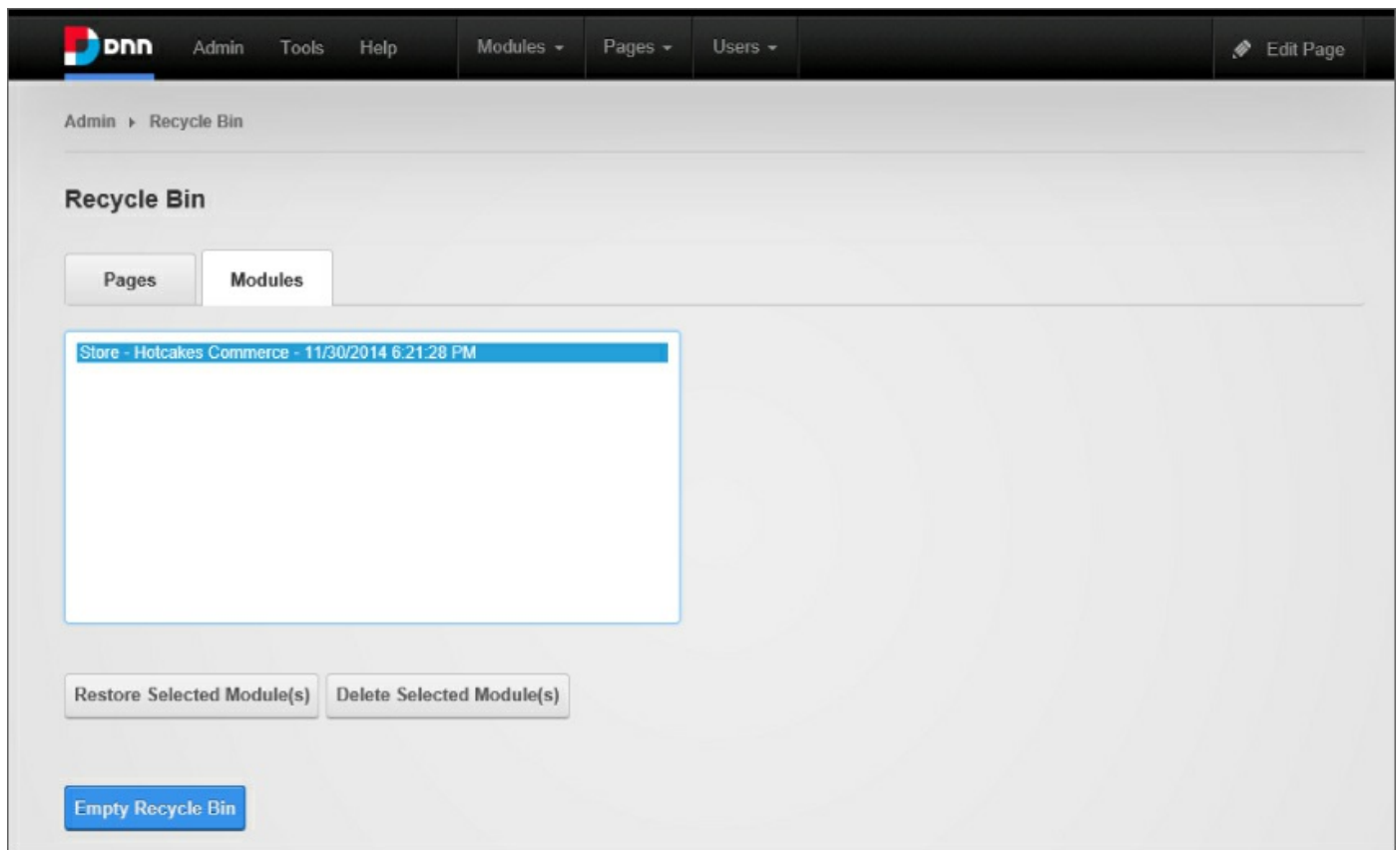


Figure 4.18

Pages are listed in a similar manner as the modules. However, the page name is preceded by an ellipsis for each level down in the site hierarchy that the page sits in the site. The page is also followed by a date, which indicates the last time the page settings were modified.

TIP

The Recycle Bin is only one of countless areas where the importance of a naming convention becomes clear. Be sure to always name everything that you create, even if the title of the object isn't immediately visible to the public. Make sure that the naming convention you use is obvious to anyone who might be managing your site.

Empty Recycle Bin

It is common for content editors to continuously create and delete pages and modules on a site. Over time, this will result in the contents of the Recycle Bin becoming quite large. While there is no built-in process that automatically empties your Recycle Bin, there is a simple button that you can click from time to time to permanently delete the contents.

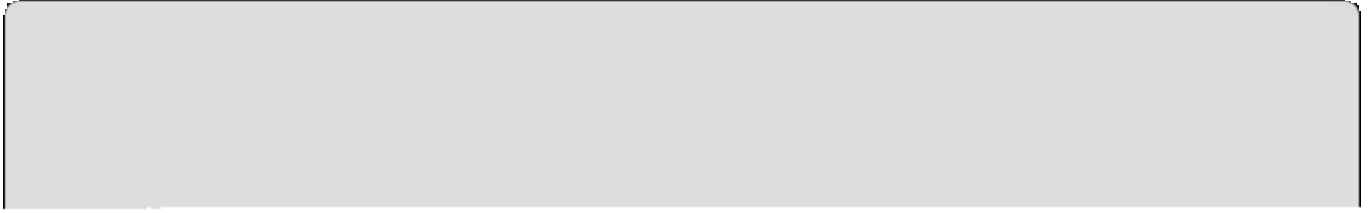
Deleting the contents of your Recycle Bin on a regular interval is considered a best practice. This ensures that the feature continues to be usable, but it also allows content editors to create new pages that would otherwise not be possible if the name matches one in the Recycle Bin. Many organizations simply send a Friday email to content editors about emptying the Recycle Bins. Unless someone responds, the administrator will empty the Recycle Bin before leaving for the weekend.

Deletion of Pages and Related Content

Pages have modules associated with them. Modules on the pages have content associated with them as well. Both modules and pages have settings that are unique to each respective instance of the page or module. When you delete a module, any content or settings associated with that module will typically be deleted as well. When you delete a page, any instances of modules and the settings for the page will also be deleted. Since the module is being deleted, its contents and settings will also be deleted.

When you delete a page by selecting it from the listing and clicking the Delete Selected Page(s) button, DNN will delete the page and any modules on that page that are not shared with other pages. If that page has any child pages associated with it—that is, any pages that sit beneath that page in the site hierarchy—you will receive a warning message and the page will not be

deleted. The warning will tell you that there are child pages assigned to this page and that those pages need to be deleted first. If you instead choose to delete everything using the Empty Recycle Bin button, all pages are deleted without prompting, including the child pages.



NOTE

The cascade effect of module contents being deleted when the respective page or module is deleted is directly dependent on the developer or vendor of that module following best practices and/or where the contents of that module are stored.

Evoq Version of the Recycle Bin Module

The Recycle Bin module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Security Roles

Security Roles in DNN are nothing more than a grouping of one or more users for a purpose. That purpose is up to you. The purpose could be to see exclusive content in a specific area of the site or on specific pages in the site. It could also be to have the ability to edit content on one or more pages or to create content in a specific module, like a blog. We could go on and on. Just remember that Security Roles can be used for content targeting, editing, and hiding sections of your site.

Security Roles work the same as most permissions-based systems, such as Windows. You first create a Security Role if it doesn't already exist to create a grouping of users. Then, you add one or more people to that group. Once you do that minimal setup, you can apply that group to various capabilities, permissions, and features.

NOTE

At the time of this writing, this is one of the few modules in the Admin menu that cannot be delegated. As a result, only administrators can manage users at this time. You will not be able to allow other Security Roles in your site to manage this feature.

Security Role vs. Social Group

As of DNN 6.2, Security Roles received some work behind the scenes to accommodate the needs of social community websites. In this kind of site, it is necessary to group people together for content-related features instead of content-editing capabilities. If there was not a distinction made in this way, your site could quickly become unmanageable and easily compromised. You would end up with far too many groups to sift through, and you would likely run into instances where groups of people were being used for permissions when that was not their intent.

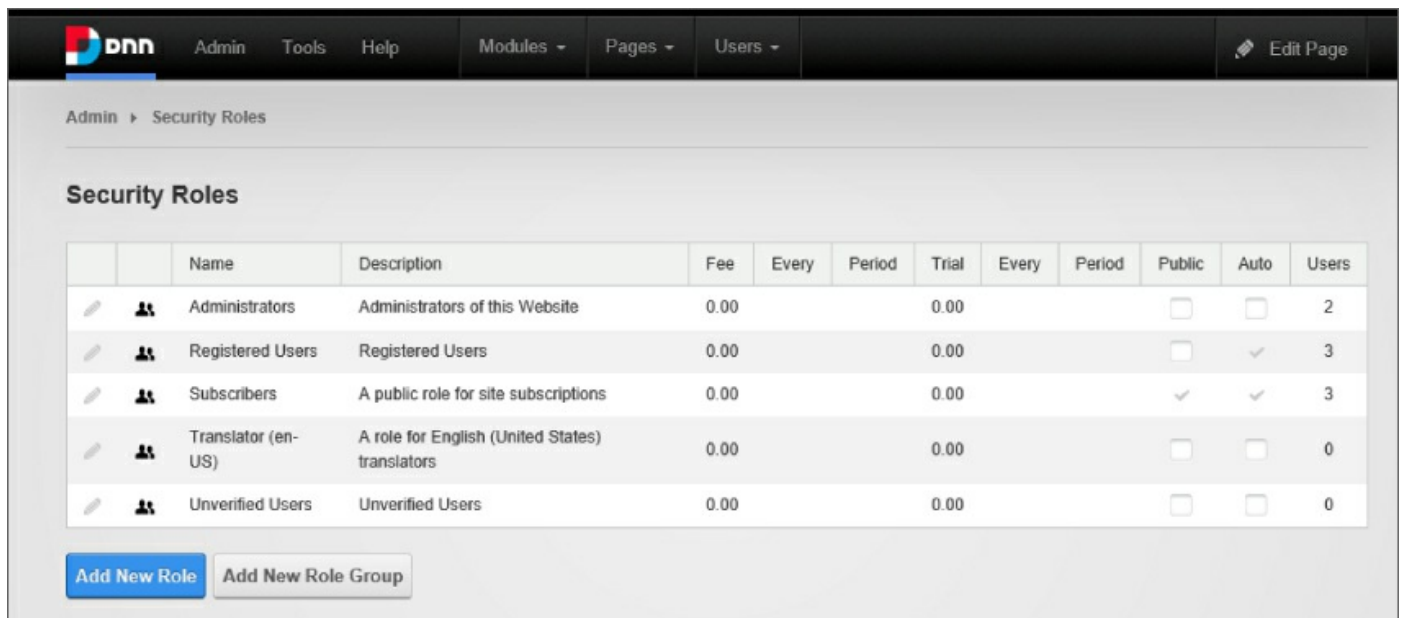
Security Roles now have a very similar sibling in DNN called Social Groups. The main difference is that Security Roles appear in the permissions grids discussed in [Chapter 3](#). Social Groups will not appear in the permissions grids and will be used only in modules that make use of them, such as the Journal, Social Groups, and Member Directory modules. Social Groups are specifically for social or community content.

TIP

If you're a DNN developer, it would be useful to know that a Social Group is simply a re-purposed Security Role. They're the same entity and have all of the same API endpoints. However, with 6.2 you now also have metadata that you can assign and reuse for development with roles like you do with module settings.

You now know that Security Roles and Social Groups each have a slightly different purpose. One gets assigned to capabilities in DNN and the other is related to social content. Both roles are a grouping of users. If you need to have any Security Role also serve as a Social Group or vice versa, a role can indeed function as both, as you'll see later in the chapter.

Security Roles (see [Figure 4.19](#)) can be found in the Admin menu under the same name, but the Control Panel also displays this same view in the Users menu as Manage Roles. Except with Social Groups, DNN will refer to a grouping of users as either Security Roles or Roles.



	Name	Description	Fee	Every	Period	Trial	Every	Period	Public	Auto	Users
	Administrators	Administrators of this Website	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	2
	Registered Users	Registered Users	0.00			0.00			<input type="checkbox"/>	<input checked="" type="checkbox"/>	3
	Subscribers	A public role for site subscriptions	0.00			0.00			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3
	Translator (en-US)	A role for English (United States) translators	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	Unverified Users	Unverified Users	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0

Figure 4.19

Built-In Security Roles

DNN comes pre-installed with a few Security Roles for your convenience. With few exceptions, these roles cannot be deleted. Some of these Security

Roles are considered default roles and others are virtual roles. The default roles allow you to add and remove users. Virtual roles are applied based on matching conditions for the user on the site. The Security Roles are explained in [Table 4.6](#).

Table 4.6 Built-in Security Roles

Name	Type	Description
Administrators	Default	The Administrators role should be reserved only for those people who you want to have full access to all administrative features on your site. Basically, they can do anything that's not reserved for the host/superuser account.
All Users	Virtual	This role is shown only in permissions grids, and it contains anyone viewing the site. If you're using this role for a purpose, just remember that it contains anonymous users. This is a role that's used for all public pages on your site.
Registered Users	Default	This role contains all users who have successfully logged in. This does not include Unverified Users. If you want to show exclusive content or sections of your site to “members” of the site, this is a commonly used role. All new user accounts are automatically assigned to this role.
Subscribers	Default	Mostly used as an example of a Security Role that doesn't necessarily need to be used for permissions-related activities. It's meant to be used together with the Newsletter module. It also shows how a role can be created for users to allow them the ability to add and remove themselves from it. All new user accounts are automatically assigned to this role. Feel free to delete this role if you are not actively using it.
Translator (en-US)	Default	This role may vary if you installed DNN using any language other than English. For example, if you installed DNN using Spanish, this role might have en-ES in its title instead of en-US. This role is

		useful if you're creating a multilingual site and is intended for those translating your content.
Unverified Users	Default	This group of users includes anyone who has created a user account but either hasn't verified their account through their email or has been unauthorized by the administrator. The Administrator can unverify this user by clicking the Authorize User button in the user's profile.
Unauthenticated Users	Virtual	If you want to target anonymous visitors specifically, this role is built for that purpose. A great example of this use might be to have one set of modules visible to Registered Users on a page and another set of modules available to Unauthenticated Users. The latter set of modules would include a value proposition of why someone should log in or create an account.

You shouldn't expect to rely on the built-in Security Roles alone. You should instead create Security Roles for each specific grouping of users for either content targeting or permissions as you see fit. The number of Security Roles that you need to have on your site will be directly dependent on the number of tasks, capabilities, targets, and experience levels of users that you need to manage. On some sites this may mean you have to create between 5–10 Security Roles. On other sites, this might end up being over 100 roles.

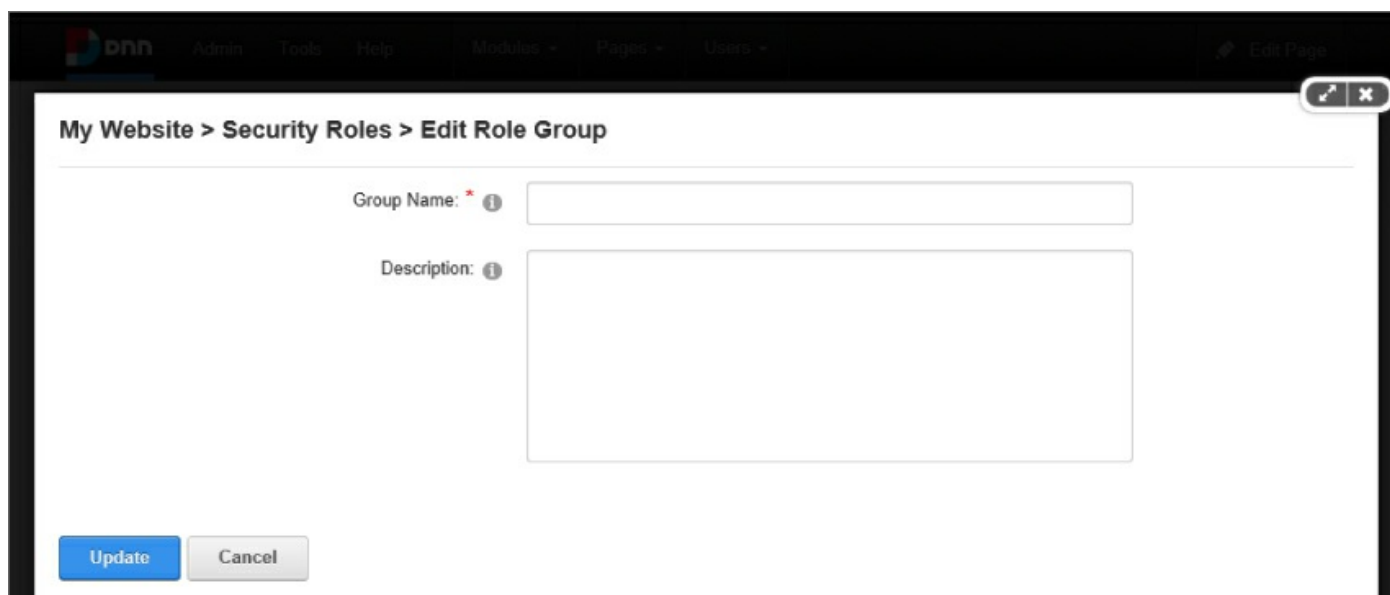
Role Groups

The previous example mentioned a site with hundreds of Security Roles. This is not uncommon on enterprise-level websites. Once you have a certain number of Security Roles, managing permissions or the roles themselves will become a cumbersome or downright impossible task. This is exactly why Role Groups was introduced to DNN. This is a feature shown only to administrators and content editors, and its only purpose is to help you organize Security Roles into categories so that they become easier to use and find. Basically, the only way you will know if a Role Group exists is if you manage users or roles or you use a permissions grid.

A Role Group should be created only if you need it. This is primarily because once you have at least one Role Group, the permissions grids and Security Roles views will all change to have a drop-down list above them to filter the

roles shown based on the Role Group selected.

Creating a Role Group is as simple as clicking the Add New Role Group button below the Security Roles grid. There are two values available to you, a Group Name and a Description (see [Figure 4.20](#)). The Name should be created in a very intuitive manner so that anyone who sees it will instantly know what the grouping is for. The description will not be seen by anyone except when in this view, so it's optional.



The screenshot shows a web interface for editing a role group. At the top, there is a navigation bar with 'onh' logo and menu items: Admin, Tools, Help, Modules, Pages, Users, and an 'Edit Page' button. Below the navigation, the breadcrumb trail reads 'My Website > Security Roles > Edit Role Group'. The main content area contains two form fields: 'Group Name' with a red asterisk and an information icon, and 'Description' with an information icon. At the bottom left, there are two buttons: 'Update' (blue) and 'Cancel' (grey).

[Figure 4.20](#)

Now that your Role Group is created, you can create any number of Security Roles for grouping.

Creating Roles

Creating a role involves looking at several more options, but is just as simple. We already discussed why a Security Role is necessary, so we will now focus on the act of creating one. You begin just like you did with a Role Group. Click the Add New Role button below the Security Roles grid.



TIP

It is very common to click accidentally the wrong button here for either purpose. For example, you might accidentally choose to create a Role Group when you intended to create a new Security Role. Just make sure that you look at which view you are working on before saving your changes.

[Figure 4.21](#) shows the edit view where a new Security Role is created. There are two sections of settings—Basic and Advanced. The Advanced Settings are rarely used and are discussed only briefly in this section.

The screenshot shows the 'Edit Security Roles' page in a DNN application. The breadcrumb trail is 'My Website > Security Roles > Edit Security Roles'. The 'Basic Settings' tab is active. The form fields are as follows:

- Role Name: Customers-VIP
- Description: Members of this role are special customers on the website.
- Role Group: < Global Roles >
- Public Role:
- Auto Assignment:
- Security Mode: SecurityRole
- Status: Approved

Buttons at the bottom: Update, Cancel.

Figure 4.21

Every role needs a name. In fact, this is the only setting that requires management when creating a new role. The Role Name will be visible to anyone who can assign permissions. This includes not only administrators but anyone who has Edit permissions to any modules, pages, or folders.

Just be sure that you institute and follow a naming convention that makes sense to everyone who manages the site when naming roles. Common conventions include prefixes to note group hierarchy and tasks. Some examples of this include *CompanyName-Marketing*, *Customers-Standard*, *Customers-VIP*, and *Bloggers*. Some characters such as spaces and periods are not allowed. The Role settings are described in [Table 4.7](#).

Table 4.7 Edit Security Roles Settings

Name	Description
Role Name	This is the name of the role that content editors and administrators will see in all permissions grids. The name should easily identify the purpose that the Security Role serves.
Description	The description is not a required field, but you should treat it as such. Over time, most sites will end up with roles that have names that appear to be so similar that content editors or even administrators won't know what they're intended for. A description will help you identify the purpose without much effort.
Role Group	The only option here will be selected for you unless you have created one or more Role Groups. By default, all roles in DNN are Global Roles until they're assigned to a Role Group.
Public Role	If checked, the role you're creating will be available for a user to add or remove from their profile. They can do this in their user profile page, if the feature is enabled. This is useful for opt-in or opt-out Security Roles, like those that you might use for newsletter email lists. The built-in Subscribers role is an example of this type of role.
Auto Assignment	If you require that users be automatically assigned to a Security Role, check this checkbox. The built-in Subscribers and Registered Users roles are great examples of this. Other useful examples that you might need in your own site include customers or community members.
Security Mode	This is a setting where you can determine what kind of role you want this to be. It can be changed after creation as well. If SecurityRole or Both is chosen, this role will appear on permissions grids. Otherwise, it will not and will strictly

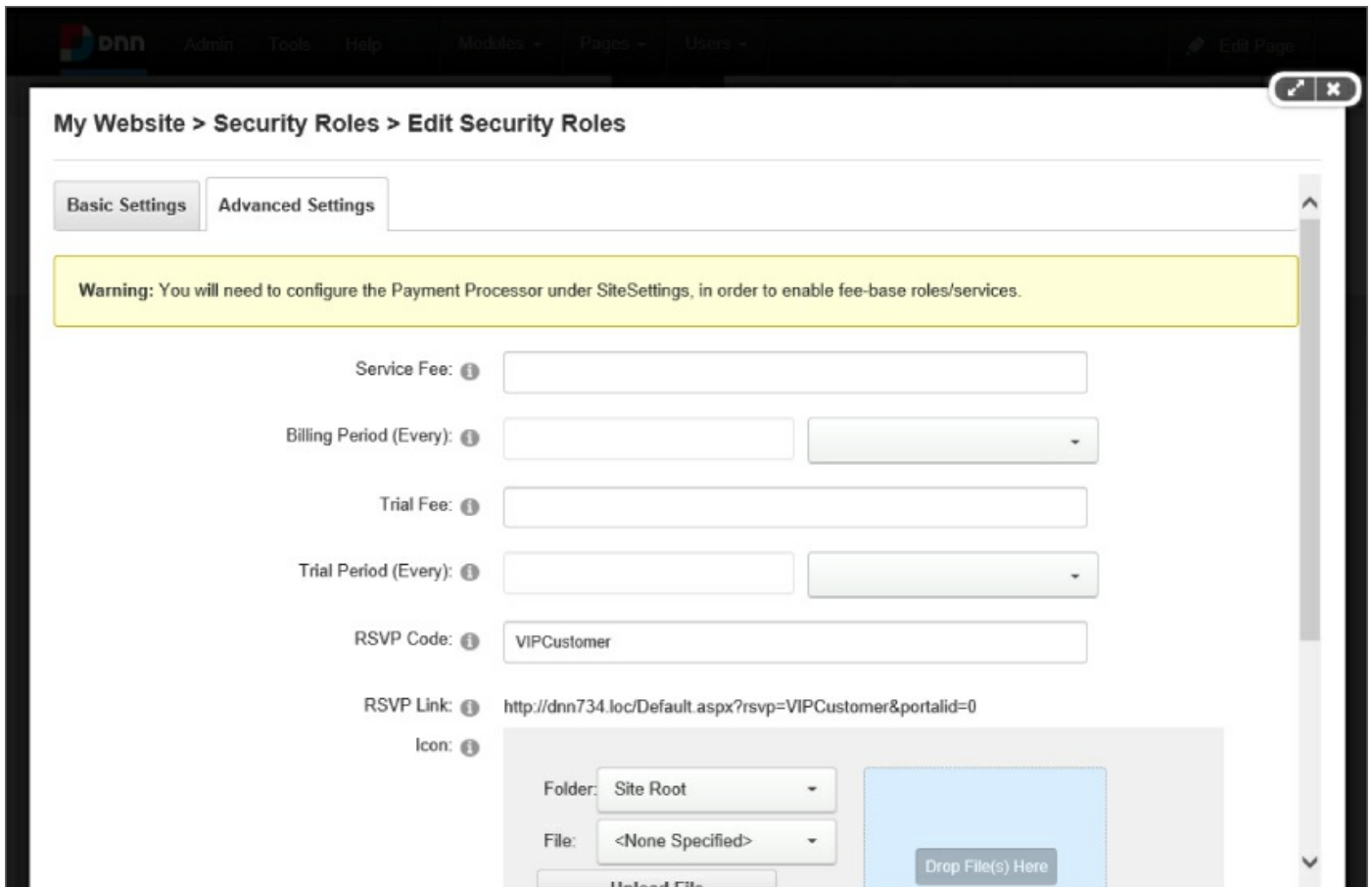
	function as a Social Group.
Status	Status is a state that only really applies to the role when it's saved as a Social Group. Social Groups can be created and require moderation before they appear to the creator. This is one way to approve the Social Group.

While in the edit view of a Security Role, you can choose Update, Manage Users in this Role, and Cancel. Clicking Update saves your changes. Manage Users in this Role allows you to choose which users are assigned to this role. Cancel disregards any changes you made and returns you to the Security Roles page.

In the case of any Security Roles you create, there is another button available, labeled Delete. This button allows you to permanently delete the Security Role. If you delete the Security Role, any areas where it is used to define content targeting or permissions will no longer respect that assignment since the role no longer exists.

The Advanced Settings tab holds two types of settings: payment and RSVP. It is possible to use the payment settings to manage the Security Roles in a way that allows you to charge a fee to be part of the role. While it is convenient that this feature is built-in, you will have a better user experience and less effort to manage this by using a third-party ecommerce module.

There is an RSVP setting in Advanced Settings that is often useful. This feature allows you to assign a code to the Security Role. When it's saved, there will be a unique URL that you forward to users that allows them to click to add themselves to the Security Role, as shown in [Figure 4.22](#). This would be useful for a situation such as adding customers to a Security Role using a link in an email.



[Figure 4.22](#)

Assignment of Roles

There are two distinct ways to assign Security Roles to users. This is one and the other will be mentioned later when we discuss user management. When you're in the Security Roles view, you use the aforementioned button when editing or adding new Security Roles or by clicking the respective people icon for the role that you want to manage the membership of. This icon will open a view similar to the one shown in [Figure 4.23](#).

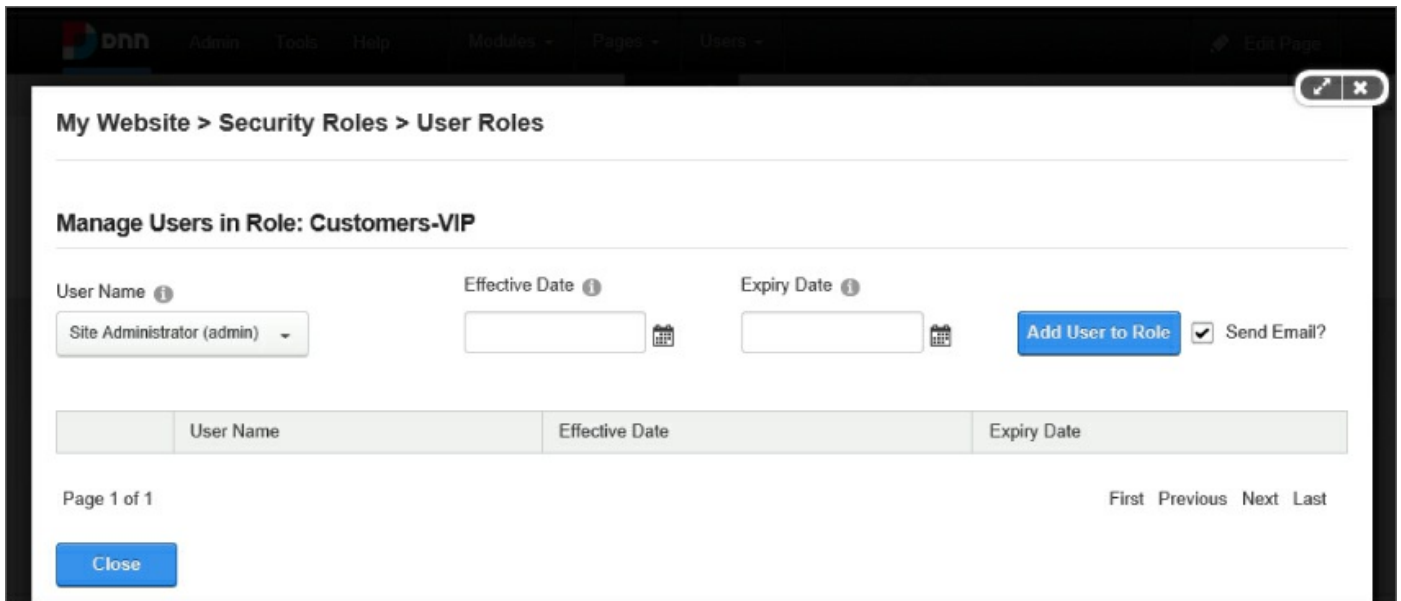


Figure 4.23

The Manage Users in Role view does a very good job of offering you quite a few options in a concise way without making it look difficult. You have only a few options, including when to be assigned to the role, when to be removed from the role, and whether or not an email is generated to notify them of their new assignment or revocation of that assignment.

The first field allows you to choose a user from the site using a drop-down list or to enter the username into a textbox for verification. It is very common on sites with a large number of members to have this view configured to use a textbox instead of a drop-down list. The drop-down list is displayed by default.

You can optionally choose to assign a date for the role assignment to become effective and determine when it should expire. If left blank, the assignment will happen immediately upon clicking the Add User to Role button and never expire. These fields are incredibly useful for many scenarios, including temporary staff, contract workers, subscription-based customers, and more.

You have a checkbox near the assignment fields that allows you to determine whether DNN attempts to email the user when you take an action. It is very common to uncheck this checkbox prior to adding a user to a role or removing them from a role. This allows you to send your own personalized email, if you wish, or simply never notify the user. Oftentimes, it's not necessary to let users know that their access to a site has been revoked.

The bottom portion of the Manage Users in Role view shows which users are currently assigned to the role, if any. As you add users in the top of this view,

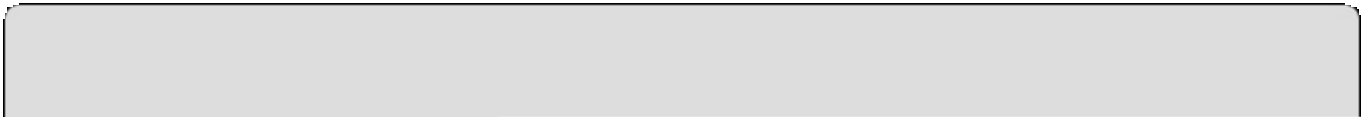
this bottom area will continue to update. Each row in the list will display the user's name linked to their profile, the expiration and effective dates (if assigned), and a link to remove the user from the role on the left side of the grid. Once you have over a certain number of users in the role, it will begin to show only a specific number per page and offer a way to navigate at the bottom of the listing. The number per page is 10 by default, but this can be changed in the module settings.

Evoq Version of the Security Roles Module

The Security Roles module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Site Settings

The Site Settings pane contains a large number of settings that you can use to perform high-level administrative configuration for your site. These settings include default metadata, user profile fields, URLs, and much more. There are too many settings to cover in this book. Therefore, several less useful or uncommon settings are skipped or only briefly covered.



TIP

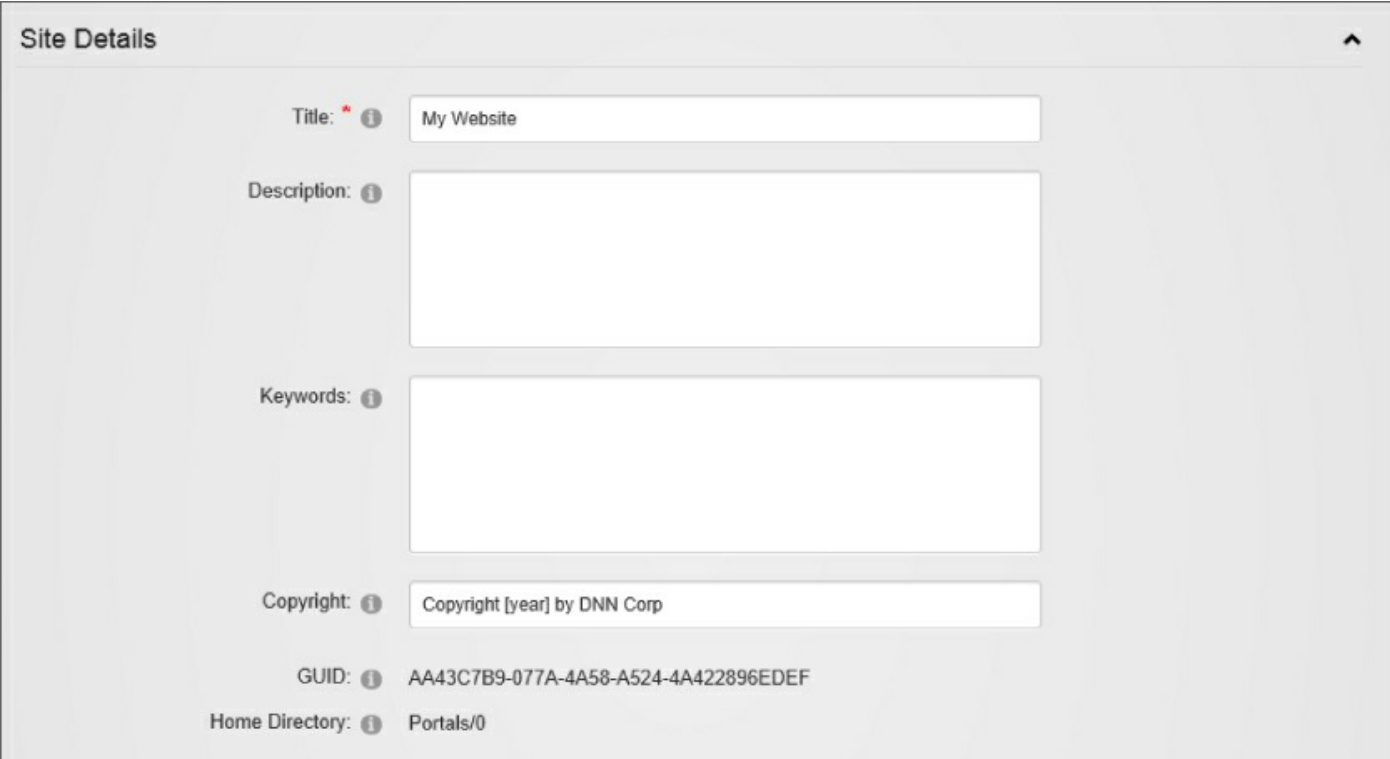
If you want a complete reference to each feature, please reference the online help section on the DNN Software website at www.DNNSoftware.com.

Basic Settings

Basic Settings mostly contains set-it-and-forget-it options that should be managed once when you build your site. Some of these settings include the default look and feel for new pages created on your site, your site logo, the copyright, and default metadata for pages.

Site Details

The Site Details page shown in [Figure 4.24](#) includes default metadata and copyright information. With the exception of the copyright, these are some of the few settings that are often managed over the lifetime of your site.



The screenshot shows the 'Site Details' configuration page. It features several input fields and informational text:

- Title:** A text box containing 'My Website' with a red asterisk and an information icon.
- Description:** A large empty text area with an information icon.
- Keywords:** A large empty text area with an information icon.
- Copyright:** A text box containing 'Copyright [year] by DNN Corp' with an information icon.
- GUID:** A text box containing the GUID 'AA43C7B9-077A-4A58-A524-4A422896EDEF' with an information icon.
- Home Directory:** A text box containing 'Portals/0' with an information icon.

[Figure 4.24](#)

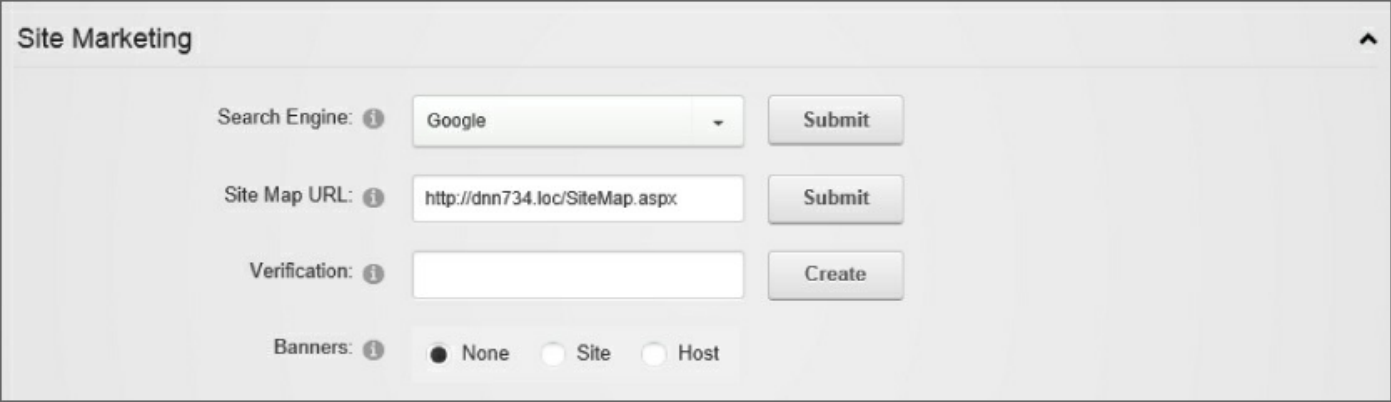
The Title, Description, and Keywords fields should sound familiar if you've managed a website for any length of time. These fields are also found in the

Page Settings for each individual page in your site. These are often referred to as metadata, or descriptive information for the page. In this case, the values are saved at the site level. This is because these values will be used in the event that you create a page that fails to have any of these values. For example, if you create a new page and forget to put in a title, the Title in Site Settings will be used on the page. This is a safeguard for those of you who need to focus on search engine optimization (SEO). These values will fill in the blanks and therefore help to prevent empty metadata from being seen by search engines.

The next setting is the copyright. This should be changed when you begin building your site, or at least before you release it. Note the `[year]` text in the default copyright. This is something that's often referred to in DNN as a *token*. This token will be replaced with the current year if your designer is using the Copyright skin object in your skin. Your site will always look fresh and updated if you use this token. You can learn more about skin objects and how they're used in [Chapter 17](#).

Site Marketing

Site Marketing allows you to manage details and actions related to getting your website discovered or managed by a major search engine. Here, you have options to submit your website to Google, Bing, or Yahoo. Submitting your site this way will simply takes you to the respective submission page on the search provider's website. [Figure 4.25](#) shows you the Site Marketing features we're discussing.



The screenshot shows the 'Site Marketing' configuration interface. It includes a dropdown for 'Search Engine' (set to Google), a text field for 'Site Map URL' (http://dnn734.loc/SiteMap.aspx), an empty 'Verification' field, and radio buttons for 'Banners' (None, Site, Host).

Figure 4.25

Every DNN site already has an XML sitemap created and dynamically updated for your convenience. XML sitemaps are used by search providers such as Google to help define the hierarchy and importance of pages on your site. If

you have a public-facing website, you really should be managing your XML sitemap already. The URL for your XML sitemap is defined in a way that allows to you to copy and paste it to a service provider or to simply submit it to the Google Webmaster Tools using the Submit button. The domain name that's used for the XML sitemap URL is defined by the Site Alias that your site is using as the default. Read more about site aliases in [Chapter 5](#) and URL management in [Chapter 20](#).

The Verification setting is specific to one of the ways that Google allows you to verify that you are the owner of a website. This is a process that you normally have to go through to use services like Google Webmaster Tools or Google Analytics. Google will give you a name of an HTML file to create on your site. Once you have that, you can paste that name into this setting and click the Create button. Your HTML file will be created, and you can then be validated as an owner of the site.

Banners is a feature that's not as well used as it could be, but it works in conjunction with the Vendors feature at the site and/or host levels. If you choose to use the banner advertising functionality in DNN, this setting helps to define the way you want your banners to load. If you choose Site, banners will be loaded from the Vendors module in the Admin menu. Choosing Host will load from the respective module in the Host menu.

Appearance

The Appearance section includes settings that help you define the default look and feel of your site as new pages are added by content editors (see [Figure 4.26](#)). These settings begin with the Logo setting. Even though it's not a best practice, a designer will include the logo as part of the design itself—known as the skin or skin package. However, if you use the Logo skin object, this setting will allow you to change the logo. This is very useful for sites or branding initiatives where the logo changes often, as the site administrator can manage the logo without having to know HTML. This is also useful for instances of DNN with multiple sites that need to reuse a design but need for the design to have a different logo on each site.

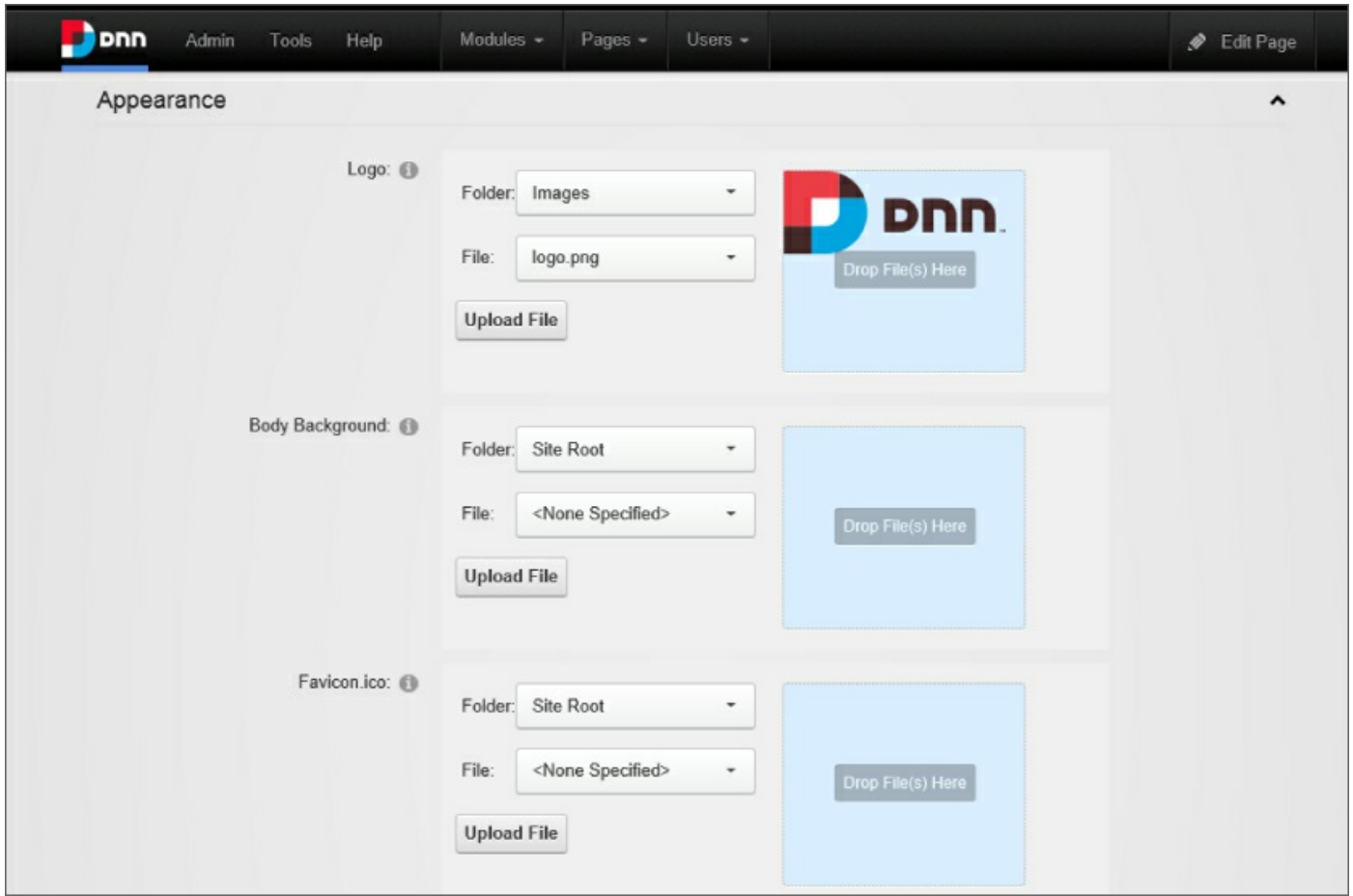


Figure 4.26



NOTE

You will see the word “skin” in all published materials released prior to this book. The word “theme” will soon replace “skin” because it's a more common and familiar naming convention for designers and administrators alike. Nothing about this feature is expected to change except for the name.

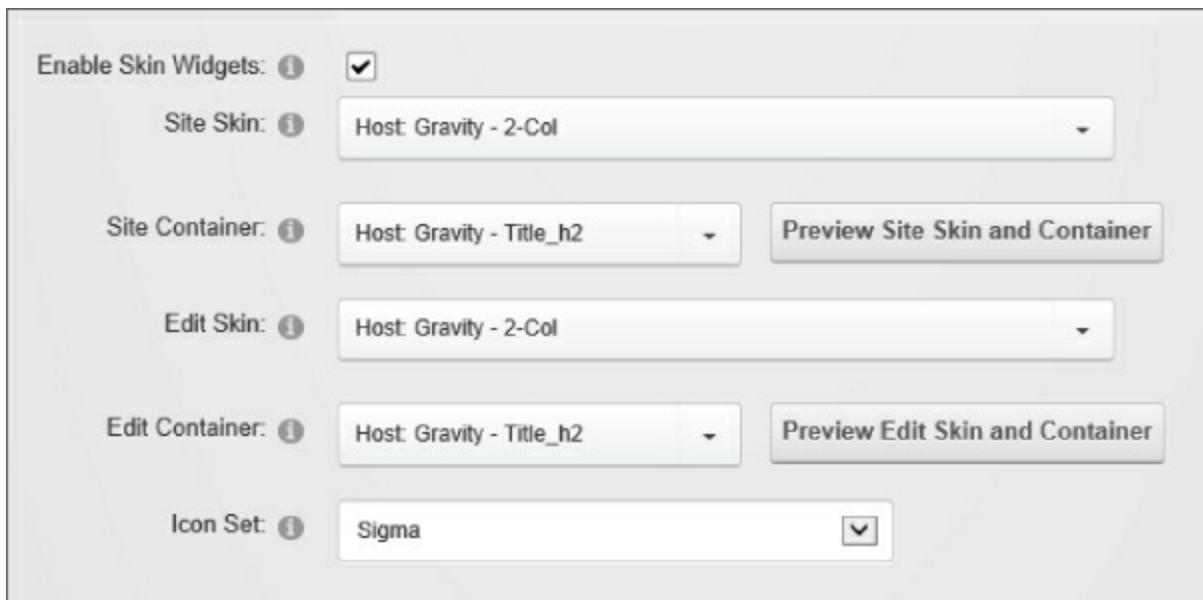
Branding on your site doesn't begin and end with the design. DNN also allows you to upload and specify a favicon for your site. Once this setting is saved, your site will display the saved favicon to users in their browser tabs and bookmarks.

Skin Widgets are client-side applications that are similar to modules but generally only consist of JavaScript and jQuery. Skin Widgets are another often overlooked extension point in DNN but can be incredibly useful. These widgets allow you to do useful things such as switch stylesheets, rotate images, relocate blocks of HTML, and much more. However, if your site is not using them, this setting should be unchecked. You might sometimes see minor performance benefits, as it means a little bit less JavaScript being loaded into the web browser.

The remaining settings of this section (seen in [Figure 4.27](#)) allow you to define the default skins and containers for your site. These default design choices will be used by pages and modules on your site, unless they override these defaults in their respective Page Settings or Module Settings. There are two types of choices shown for your skins and containers, Site and Edit. The skin and container chosen for Site will be applied to all pages that do not define their own defaults, as previously noted. The Edit skin and container will be used when you enter into an edit view where all of the modules but one is removed from the page. For example, some modules will enter into an edit view when you choose to edit settings or manage its content. Skins and containers will be explained in more detail in [Chapter 17](#).

NOTE

This edit view is more commonly referred to as module isolation for developers. It is a common way for developers to show only the edit options or features for their module. The alternative to this view is the pop-up view, which is the default. When a pop-up is used, the skin and container settings are not used, as they are no longer necessary.



The screenshot displays the 'Advanced Settings' section of a site configuration interface. It features several settings:

- Enable Skin Widgets:** A checkbox that is checked.
- Site Skin:** A dropdown menu set to 'Host: Gravity - 2-Col'.
- Site Container:** A dropdown menu set to 'Host: Gravity - Title_h2', accompanied by a 'Preview Site Skin and Container' button.
- Edit Skin:** A dropdown menu set to 'Host: Gravity - 2-Col'.
- Edit Container:** A dropdown menu set to 'Host: Gravity - Title_h2', accompanied by a 'Preview Edit Skin and Container' button.
- Icon Set:** A dropdown menu set to 'Sigma'.

Figure 4.27

Advanced Settings

The Advanced Settings section contains settings that are not commonly changed. In fact, more so than the rest of Site Settings, the settings on this tab really are considered to be set-it-and-forget-it settings. This section of settings includes system page management, administrator assignment, and usability settings.

If you were logged in using a host/superuser account, the Advanced Settings tab also includes sections that are only shown to a host/superuser. Those settings are explained in [Chapter 5](#).

Page Management

The Page Management section allows you to set system pages, as shown in [Figure 4.28](#). The specified pages are used by DNN to help properly navigate

users to the right pages when certain system actions take place. Such actions could include clicking the site logo, logging in, registering, or simply performing a search. These pages tell DNN where that functionality exists to ensure a consistent user experience for your site visitors.

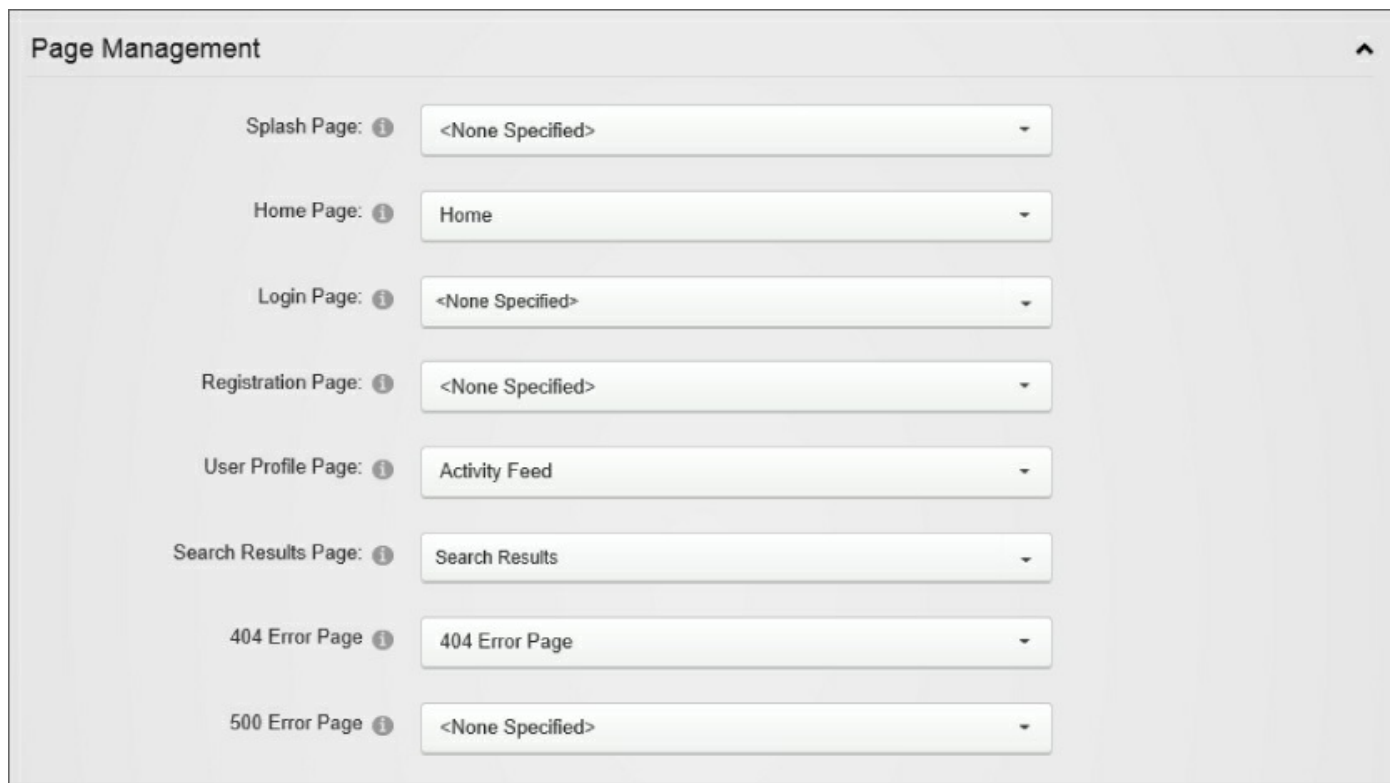


Figure 4.28

[Table 4.8](#) describes the Page Management settings and specifies as well the pages that are assigned to these settings when DNN is first installed. These pages might not still exist on your current DNN site.

Table 4.8 Page Management Settings

Setting	Default Page	Description
Splash Page	<None Specified>	Splash pages really aren't used much anymore, but this setting will allow you to intercept a user before they arrive at your home page the first time to show them a different page. This could be useful for things like product sales, value propositions for user registration, and more. However, most visitors find the splash page an annoyance.

Home Page	Home	This setting tells DNN which page should be the home page. This page will be shown when the site logo is clicked and when someone visits the root URL of your site, such as www.domain.com . If, for some reason, this setting is left as <None Specified>, the first page in your site hierarchy will be used.
Login Page	<None Specified>	DNN technically doesn't have a login page out-of-the-box. It's dynamically generated for you. If you want to change this to have a specific page for login to ensure a secure login or to wrap the login with benefits, advertisements, or value propositions, or you want to use a third-party login module, this setting will be very useful.
Registration Page	<None Specified>	Works the same way as Login Page, only it's used to register new user accounts instead of accepting login information. This also is a useful setting if you are using a custom registration module instead of the default registration process.
User Profile Page	Activity Feed	This page is where users are sent if they choose to view any user profile, including their own. This page generally should have at least one View Profile module on it. This module allows the user to edit their profile. A profile page is created for you by default. It is not considered a best practice to create your own user profile page except in instances where they didn't already exist.
Search Results Page	Search Results	DNN comes to you with a robust search engine that will index the content of your site. The default design will also include a search box. This setting will determine which page on your site a user will be sent to when executing a search. The landing page chosen here should have a Search Results module on it.
404 Error Page	404 Error Page	This is the page that all traffic will be sent to when the page is either not found or, in some cases, when the page is not accessible to the visitor due to

		permissions.
500 Error Page	<None Specified>	When unrecoverable errors are experienced, you can send the visitor to this page to give a more user-friendly user experience. One example for this might be to have a comment or survey to get more information about how the error might have happened.

Security Settings

The Security Settings section allows you to designate a specific administrator account as the administrator as well as determine whether you want to hide the login link from users (see [Figure 4.29](#)).

The screenshot shows a 'Security Settings' panel. At the top left is the title 'Security Settings' with an upward-pointing arrow on the right. Below the title, there are two settings. The first is 'Administrator:' followed by a help icon and a dropdown menu currently showing 'Site Administrator'. The second is 'Hide Login Control?' followed by a help icon and an unchecked checkbox.

[Figure 4.29](#)

There can be many user accounts with administrative rights on your site, but only one can be considered to be the administrator. This setting allows the site to know which user account to use for high-level tasks. Probably the most significant purpose behind using a specific account as the administrator is that emails sent by DNN from your site will use the email address assigned to this user account as the From address in the email. This means that you want to be aware of which email address this account uses. Anyone who chooses to reply to an email from your site will respond to that email address.



NOTE

You definitely want to be aware of the email address of your chosen administrator, but you should probably also consider the Username and Display Name of the user account that's chosen as the administrator. Either value could potentially be visible to website visitors at various times of administration and moderation.

The Hide Login Control setting is pretty self-explanatory. The Login link will no longer be visible on your site. This is useful for many websites, but it should be noted that this does not prevent anyone from logging in completely. If the user can discover the login URL in some way, they will still be able to log in.

TIP

Completely removing the ability to log in is outside of the scope of this book, but it is a moderately advanced task that can be achieved. Some uses cases would be unique in this regard, so it is recommended that you seek guidance for this in the DNN forums to learn how to do this for your specific site at www.DNNSoftware.com/forums.

Payment Settings

The payment settings found in this section allow you as a site owner to accept payment to have access to Security Roles. This is useful for sites that might have exclusive content. You can save your payment gateway information here and then allow users to subscribe to the content, which will result in them being added to one or more Security Roles. These same Security Roles should then be used for view permissions on the modules or pages that are part of the exclusive areas of the site.

It is worth noting that while this capability is built-in and it works, there are a number of third-party extensions that work very well to provide this functionality in ways that will be easier for you, your team, and your customers to manage. The third-party extensions will also support more than one payment provider. DNN ships only with a PayPal payment provider.

Usability Settings

Usability Settings allow you to adjust some of the ways that DNN presents itself to visitors, content editors, and administrators. Usability goes beyond things as obvious as pop-up windows. It includes the Site TimeZone setting shown in [Figure 4.30](#).

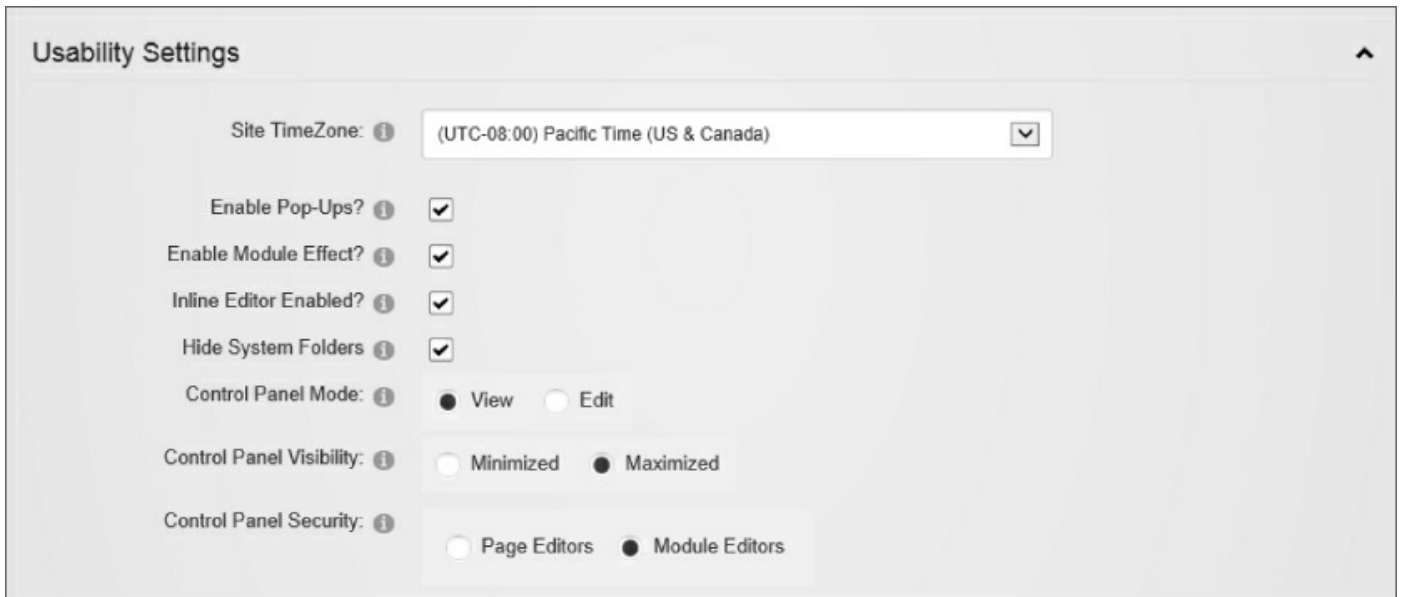


Figure 4.30

DNN was originally born as a universally adopted web application framework in most countries around the world. It has a high adoption rate in Europe, South America, Australia, and of course North America. As such, it incorporates a lot of different languages and time zones. The Site TimeZone setting allows you to set the time zone of the site, as it clearly states. DNN will store times in the database using UTC or the Universal Time Coordinated format. The time zone setting allows you to set the time zone so that times presented in areas such as the Event Viewer are relative to your location, even when your web server is in another time zone.



NOTE

Dates are not actually stored in UTC in many areas of the database at the time of this writing. However, this is something that is actively changing and should be complete soon.

The rest of the usability settings revolve around how DNN might react to actions users take, primarily as a content editor or administrator, with a couple of exceptions.

If you choose to uncheck the Enable Pop-Ups setting, it will disable pop-ups from appearing throughout DNN, except for the case of confirmation prompts and when third-party or custom modules might explicitly create pop-ups.

DNN has an inline editor that is primarily present for module titles and the HTML and HTML Pro modules. This setting is enabled by default, and it will allow you to perform quick edits to the module title and HTML-based content. This is convenient for many users, but on some sites you might want to disable this feature. Disabling this feature could increase usability in some cases, and it would reduce the number of resources loaded for content editors, resulting in minor page load improvements. Disabling this setting would not result in faster page loads for other kinds of site visitors, though.

NOTE

You might see the HTML module referred to as the Text/HTML module in older documentation and community articles. The HTML and HTML Pro modules are the same module, except that the HTML Pro module is found only in the commercial editions of DNN. Both modules work the same way, but HTML Pro includes more features, such as content workflow, auto-save, content versioning, and side-by-side version compare.

The remaining usability settings are present for backward compatibility for those sites that are using an older Control Panel version.

User Account Settings

The User Account Settings area consists of all of the high-level settings that will help you adjust the login, registration, and profile functionality of your site for your visitors. Over the last few years, this area has been enhanced a great deal to put all of these settings in one place and to make it flexible enough to serve most registration or login scenarios without having to customize an underlying provider.

NOTE

This section alone could easily fill its own chapter. In the interest of brevity, only the most common and useful settings are discussed. You can discover more about the remaining settings by visiting the DNN Online Help at www.dnnsoftware.com/Help.

Registration Settings

The process of creating a user account is usually referred to as registration. The Registration Settings shown in [Figure 4.31](#) govern the way new users join your website. It should be immediately obvious when looking at the number of possible configuration options that you can customize the registration process in a multitude of ways.

The screenshot displays the 'Registration Settings' page in the DNN administration interface. The settings are as follows:


- User Registration:** Radio buttons for None, Private (selected), Public, and Verified.
- Receive user registration notification:** Checked checkbox.
- Use Authentication Providers:** Unchecked checkbox.
- Excluded Terms:** Empty text input field.
- Use Profanity Filter:** Unchecked checkbox.
- Registration Form Type:** Radio buttons for Standard (selected) and Custom.
- Use Email Address as Username:** Unchecked checkbox.
- Require Unique Display Name:** Unchecked checkbox.
- Display Name Format:** Empty text input field.
- User Name Validation:** Empty text input field.
- Email Address Validation:** Text input field containing the regex: `^[a-zA-Z0-9_+%&*/=-^{}~](?\.?[a-zA-Z0-9_+%&*/=-^{}~])*\@[a-zA-Z]`
- Use Random Password:** Unchecked checkbox.
- Require Password Confirmation:** Checked checkbox.

Figure 4.31

The User Registration setting is perhaps the most common setting used when visiting this area. It allows you to change the workflow used to onboard a

visitor into their new user account on your website. You have four options, described in [Table 4.9](#).

Table 4.9 User Registration Options

Option	Description
None	Visitors to your site will not be allowed to create a new user account in any way. Only administrators can create new user accounts. This will remove the login link from all DNN skin objects and will even redirect the users if they happen to know the original registration URL. This is the most common option for private sites, or sites that are using alternative authentication providers such as Active Directory.
Private	This is the default workflow that is enabled on a new installation of DNN. When selected, new user accounts must be approved by an administrator in the Admin  User Accounts or Manage Users area of the Control Panel. This is the most common option for intranet and extranet sites that use the default authentication provider.
Public	This registration workflow should not be used on most sites. There are no features that allow you to verify the user in any way. Once the user account is created, the user is immediately logged in. This often results numerous accounts created by spammers and results in many more support requests related to non-verified email addresses.
Verified	The most commonly used option for public-facing websites. This option allows users to create their own user accounts, but they cannot log in until they verify their email addresses. There will be an email sent to them with verification instructions. This option does not prevent spam accounts from being created.

Use Authentication Providers

Many sites will want to provide more than a single way for a new user to log in and register on your site. This is especially popular since major social sites like Twitter and Facebook have made login mechanisms that any site owner can implement with a bit of code. You have these options available to you out of the box with Twitter, Microsoft Live, Google, and Facebook options available for host/superusers to install. They are not installed by default. See

[Chapter 5](#) for more information about these additional providers.

If you have any additional authentication providers installed and enabled, this feature will display those other options to users on the login form when this checkbox is checked.

Excluded Terms

Your site might need to prevent new users from creating user accounts with certain terms in its username. This is very common and a best practice for sites with a high volume of users, social features, or any other branding concerns. Imagine if your site is being implemented for a brand as recognizable and valuable as Microsoft. You might want to prevent users from including product or trademarked terms like “Microsoft” or “Azure” in their usernames. To do this, enter each of the words you want to prevent from being part of the username, separated by a comma. In this case, you would use `microsoft,azure`.



TIP

You should be careful when using this feature, as it might cause unintended consequences. Avoid using common words and words that might get formed incidentally from two other words being side by side. One final note is that if you're too restrictive with settings like this, you could also have an unintended side effect of preventing site registrations altogether.

Use Profanity Filter

This checkbox allows you to filter out profane language from usernames. Essentially, when enabled, the users cannot create an account with any profane words in it. Checking this checkbox alone will do nothing. You also need to define the profane words that you want to prevent from appearing in a username. This is done in the Admin \triangleright Advanced Settings \triangleright Lists feature. This feature is discussed later in this chapter.

Registration Form Type

You've been using standard registration forms if you've been using DNN at all. This option simply displays the default fields and options for a new user to fill in when they want to create a user account on your site. The default form is somewhat verbose for most sites. It might be a good idea to consider a custom registration form, which allows you to specify the fields you want to show. [Figure 4.32](#) shows what the difference could be when using this option along with one of the next settings, Display Name Format.

Figure 4.32

In the registration form on the left, you can see that there are five different fields that have to be successfully filled out in order for the user to simply proceed to the next step. If you've ever performed usability testing, you probably already know that the success rate of users getting past this view is not optimal.

Now compare the registration on the left to the one on the right. This example combines the Custom Registration Form Type with the Display Name format to require only three pieces of information for a new user to create their user account.

Use Email Address as Username

If you want to remove the option of users creating their own username or you simply want to make it easier for users to log in and remember their username, this option is one of the most ideal ways to achieve those goals. It greatly reduces support issues related to user registration and authentication. If you want your users to use their email address as their username, this option should be used instead of trying to do the same thing with the Custom Registration Form Type.

When using this feature, it is most ideal to only be used on new websites. Existing sites will have a slight usability issue, in that existing users will still

use their existing username and not their email address. This could cause confusion and support issues. In addition, since the email address would be the username, email addresses would need to be unique on the site.

Require Unique Display Name

Display Name is the primary field that nearly all modules use when displaying user-generated content in places like the Journal, blog comments, and social groups. The Display Name can be duplicated by default. In sites that have social features such as the previous examples, it might be a good idea to require a unique Display Name. If enabled, the Display Name will be a required field on the user registration form. Anyone who attempts to create a user account with an existing Display Name will be asked to create a different Display Name before their account can be created.

Display Name Format

Every user account has a Display Name as discussed in the previous setting. This is the chance for the user to create a potentially unique way to express themselves to others on your site for personal branding purposes or to convey their personality. On many sites, such a field can be very important to the user's online identity. If you want users to be able to express themselves in this way, leave this field empty. Otherwise, you can enter a value known as a token to define a format for users. If you enter anything into this field, DNN will attempt to use it, and the Display Name will no longer be an option on the registration and profile pages. [Table 4.10](#) outlines the tokens that this setting accepts.



TIP

If at any point in your site's existence, you decide to use this feature, attempt to do so at an off-peak time. Changing this setting will cause DNN to iterate through each user account and update it individually. On a site with a large number of users, this could potentially cause the site to load slowly until all users are updated.

Table 4.10 Display Name Format Tokens

Token	Description
[USERID]	Replaced by the user account's unique UserID number
[FIRSTNAME]	Replaced by the First Name defined in the user's profile
[LASTNAME]	Replaced by the Last Name defined in the user's profile
[USERNAME]	Replaced by the username chosen by the user

Use Random Password

As of DNN 7.1, this setting should no longer be used. This setting was useful only when a password could be sent to a user during the password recovery process. Password recovery is no longer supported in DNN due to the security standards defined by the DNN Corp. security team. You can only reset passwords now, so this setting will actually prevent users from being able to log in until they reset their passwords.

Require a Valid Profile for Registration

You'll find that this setting is disabled by default on new installs of DNN. This is for very good reason. Most sites have at least a few fields in the user profile that are required. If you checked this setting, all of those required fields would need to be properly filled out prior to the user being allowed to complete the registration. Having this option enabled would have a direct impact on how many people you lose during the registration process. This setting should be used only on sites where your users have no choice but to register, such as partner or affiliate programs, intranets/extranets, and some membership groups.

NOTE

DNN's default settings will force users to fill out any required fields in their profile when they come back to log in on your site. Keeping this setting unchecked will allow you to capture your user's registration information, without discouraging them from completing the registration.

Redirect After Registration

When a new user registers on a DNN site, the user will be sent back to the page where the registration link was clicked. If you want the user to be sent anywhere else, select that page here. This setting can become a very interesting option if you were to combine it with redirection settings in the Page Settings, a content-targeting redirection module from the DNN Store, or content-targeting through security roles on the resulting page. This would allow you to route new users through some kind of additional workflow or to customize a thank you or other value proposition.

Read-Only Settings

There are quite a few settings that are presented to you in a read-only way, meaning that you cannot edit them. These settings are kept in a configuration file that only a host/superuser can edit. This kind of protection is required because changing these features could result in a security risk, a broken website, or a user experience that would deter potential and existing users from coming back. In some cases, ASP.NET prevents even DNN from changing the setting in a view like this. [Chapter 9](#) discusses these settings in more detail.

Login Settings

The last section clearly allowed you to configure how new users are onboarded. This section allows you to configure the user experience while an existing user is attempting to log in on your site. Unlike registration, there are a minimal number of settings to configure.

There are two settings that allow you to enable CAPTCHA—for associating logins and for retrieving the password. Associating logins is the process when you have two or more kinds of authentication enabled and you want them to

be associated with each other. In either case, CAPTCHA can help improve the security of your site and prevent spammers.



NOTE

CAPTCHA was a setting in Registration Settings as well, although not discussed. This acronym stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart. It is that generally annoying feature that asks you to enter the letters and numbers inside of an obscured image. As annoying as it may be to you, it can be highly effective at preventing malicious programs (bots) from attacking websites.

Requiring a valid profile for login is a default setting, and it is good to enable it throughout the lifecycle of your site. This forces users to fill in any required profile fields following registration or at any point if you determine that you need to introduce a newly required profile field. The user needs to save a valid value before being allowed to complete the login process.

The redirect settings work the same way as the setting discussed in the previous section, but for two distinct login actions, logon and logoff. The default behavior for login redirect is to send the user back to the page where the login link was clicked. The default behavior is a bit more complex for logoff. DNN will attempt to send the user back to the page where the logoff link was clicked. However, if users are not allowed to view that page for any reason (such as due to permissions), they will instead be sent to the home page. For either setting, you will be able to change this default behavior.

Profile Settings

Profile Settings exposes one of the more powerful and under-appreciated features that DNN has to offer. The Profile Settings expose to you a few features, but the primary feature is the Profile Editor, which allows you to customize completely the profile fields that a user account has in nearly every way possible. This is the backbone of all of the user profile features in DNN.

Vanity URLs

Vanity URLs are somewhat new to DNN but are a very welcome feature that improves the user experience on sites that are trying to build a community of users. Members of your site now have a user-friendly and personalized way to share their profiles with other users either on the same site, or elsewhere if

you're building an external community. Vanity URLs are discussed in [Chapter 21](#), but we'll review the features shown in [Figure 4.33](#) in this section.

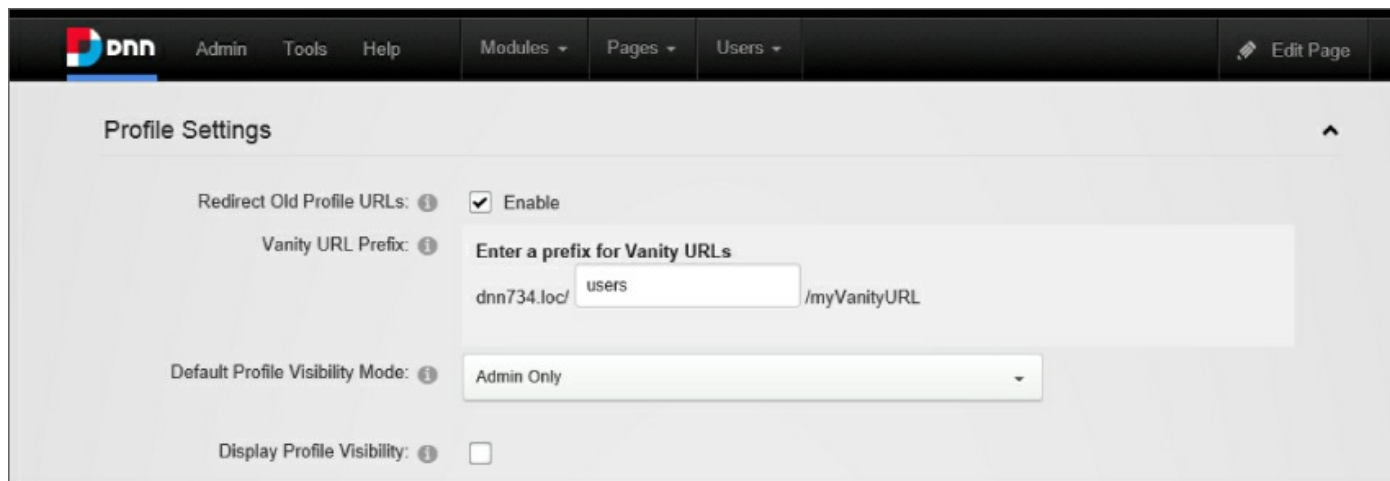


Figure 4.33

The first option allows you to redirect old profile URLs. This is very useful if you're upgrading from an older version of DNN that didn't have vanity URLs as an option. This allows users to be redirected to the right page if they have an old link or bookmarked profile page.

Your next option is the vanity URL prefix. This allows you to customize the value that becomes the folder path after your domain name. This path will be “Users” by default, but this option allows you to customize it. Some obvious examples include Fans, Customers, or Members, but imagine having a community that has a specific name for its members. You may have noticed the term DNNizen on the [DNNSoftware.com](http://www.dnnsoftware.com) site. If desired, this could be a term used in this setting (but it isn't). If a user were named Jane Doe and she used a URL associated with her name, the resulting URL might look like this:

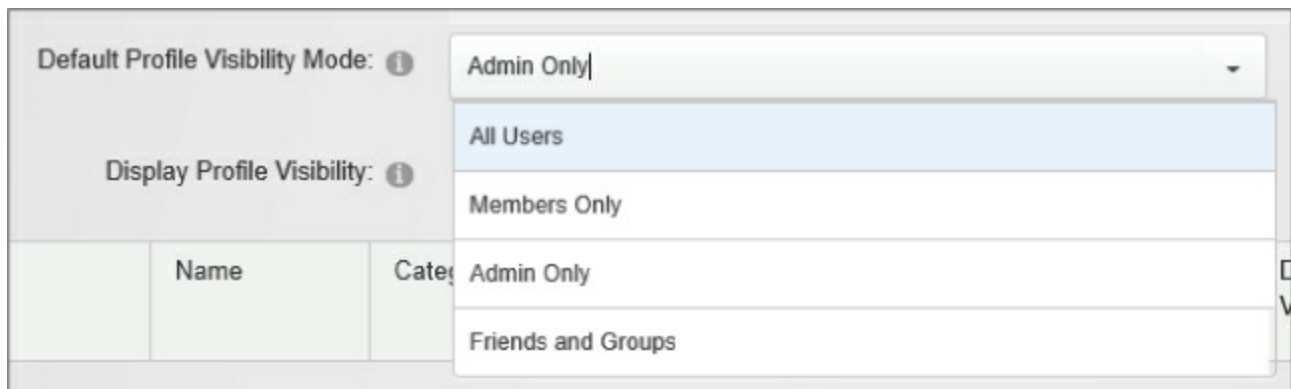
<http://www.dnnsoftware.com/DNNizen/Jane-Doe>

Profile Visibility

Profile visibility is likely to be a feature that is important to any DNN site that allows visitors to have their own user accounts in any way, especially for user content generation features, such as blogs, comments, and other social features. When these kinds of features are present, their profiles will be shown to other visitors as they contribute content. Therefore, you might want to know about and determine whether you want to make adjustment to the profile visibility settings on your site.

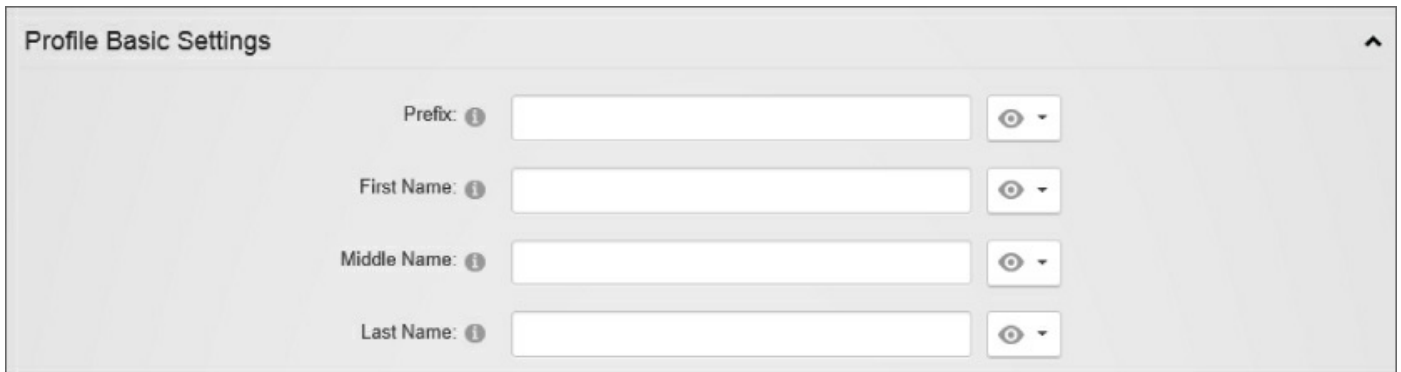
The first of two settings that we'll talk about is Default Profile Visibility

Mode. It has a drop-down list where you can decide between four different options. Each of them should be self-explanatory, as their labels are pretty well defined (see [Figure 4.34](#)). When you choose any of these options, you are deciding the default setting of the visibility of all profile properties for members of your site. This will not override any values in the grid below it, nor will it override the values that a user might change in their own profile. For example, in a clean installation, you can change the default of Admin Only to anything else, and if a member had changed the value of a field from Admin Only to Members Only, their setting will override yours. This highlights an important point of managing a DNN site with a membership of users: have a plan before you release your site so that you can hopefully get this setting right the first time, if possible.



[Figure 4.34](#)

The second setting we'll discuss is called Display Profile Visibility. This feature is checked by default. When checked, it allows the members of your site to be able to override the privacy settings of any individual profile field. This is useful if you want to allow this level of granular control to be given to users. However, in practice, this can potentially create a lot more work for site administrators, designers, and module developers since they cannot know for sure which profile settings will be visible. Unless you have legal obligations to do otherwise, you may want to instead set this at the administrator level and then make it clear to your members what fields will be shown to other members or to the public. [Figure 4.35](#) shows the icon that is displayed next to profile fields when this feature is enabled.

The image shows a screenshot of a web form titled "Profile Basic Settings" in the top left corner. The form contains four input fields, each with a label and a small information icon (i) to its left. The labels are "Prefix:", "First Name:", "Middle Name:", and "Last Name:". Each input field is a simple white rectangle with a light gray border. To the right of each input field is a small square button containing an eye icon and a downward-pointing arrow, indicating a visibility or dropdown menu. The entire form is set against a light gray background.

[Figure 4.35](#)

Profile Editor

The Profile Editor is the last setting in this tab that we'll discuss. It does exactly what its name suggests. It allows you to edit features of profiles on your website. Using this editor, you can create profile properties of nearly any kind, to be used with all kinds of profile scenarios and integrations. Profile properties support a wide array of options, including multiple languages (or localization), validation, and more. You can also group together the profile properties so that they are shown to website users in a way that's better for usability.

You'll notice that there are a handful of features of each profile property, as shown in [Figure 4.36](#). First, you can edit any of them by clicking the edit icon. When you do, you'll see a pop-up appear if it's enabled or be taken to a new view altogether. This allows you to change all aspects of the profile property, including the grouping, how it's named, how it's validated, how a user interacts with it, what kind of field it is, and more.

























































	Name	Category	Data Type	Length	Default Value	Validation Expression	Default Visibility	Required	Visible
   	Prefix	Basic	Text	50			All Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	FirstName	Basic	Text	50			All Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	MiddleName	Basic	Text	50			All Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	LastName	Basic	Text	50			All Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Biography	Basic	RichText	0			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Photo	Basic	Image	0			All Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Telephone	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Cell	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Fax	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Website	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	IM	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Twitter	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	Skype	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>
   	LinkedIn	Contact	Text	50			Admin Only	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 4.36

Next, you can delete most of them. If you choose to do this, you'll be able to re-create the profile property if you want, but the real reason you might want to do this is to reduce the number of fields available to website users, thus encouraging more complete profiles.

The profile properties that do not have a delete option are those that are directly tied to authentication and/or overall site membership. Deleting them would cause some features to either not work or become unusable, which is why you are not given the opportunity to delete them.

Perhaps in an update to this feature, you'll be able to drag and drop each profile property to re-order them as you desire, but until then you'll find up and down arrows that allow you to re-order by clicking.

The last two features that are editable through this view include the Required

and Visible columns. These settings have a checkbox associated with them and allow you to turn the feature on or off. These each can be set when editing the profile property as well.

In the case of Required, this allows you to define whether the profile property is required to have a valid value. When enabled, this option can work together with the features in the Registration and Login Settings sections to ensure these values are populated.

Visible simply allows you to define whether the profile property is visible to users or only to administrators. Having hidden profile properties is useful to keep information about members of your site, without them being able to see or edit them. A great example of this might be a “Report To” profile property in an intranet where you select the manager that a user reports to.

Creating New Profile Properties

When you decide to create a new profile property, many of the fields and values available should be self-explanatory, as you will walk through a short wizard-based edit view. You simply need to click the Add New Profile Property button at the bottom of the profile properties grid. Each step in the profile property wizard has all of the same fields that will be present if you were editing an existing profile property.

You might want to consider having another web browser tab open that has an existing profile property open in its respective edit view for comparison when creating your first profile property.

One of the things to note about creating or editing of a profile property is that you'll have a list of 17 options to choose from when deciding what the data type is, as shown in [Figure 4.37](#). The data type will determine how the profile property is displayed to members of your website. For example, if you choose Page, they will be shown a field that allows them to select a page from the site. If you choose Date, they will be given a field that allows them to select a date from a date picker, enter one manually, and so on.

Add New Property Details

The first step in setting up a Profile Property Definition is to define the property's details. Enter the details in this page and click "Next" to create the Property Definition. **Note:** All fields marked with a red asterisk are required.

Property Name: *

Data Type: *

- Checkbox
- Country
- List
- Locale
- Page
- Region
- RichText
- TimeZone
- Integer
- Multi-line Text
- Text
- TrueFalse
- Image
- TimeZoneInfo
- Date
- DateTime
- Unknown

Property Category: *

Length:

Default Value:

Validation Expression:

Required:

Read Only:

Figure 4.37

Stylesheet Editor

You might have already guessed, but the Stylesheet Editor allows you to enter and maintain CSS (Cascading Style Sheets) code on your website. When you install DNN or create a new site as a superuser/host, this field will contain some placeholder CSS.



NOTE

How to write CSS is beyond the scope of this chapter. [Chapter 17](#) covers CSS briefly, but if you're really interested in learning CSS, a great online resource is www.w3schools.com/css.

If you were to enter valid CSS into this field and save it, a file will be created in the respective site folder for the current website, called `Portal.css`. This is the last CSS file that will be loaded into the HTML on page loads, which should theoretically allow your CSS to override any other CSS in the site. While this sounds convenient and powerful, you should avoid using this feature at all costs. There is potential for your changes to be completely deleted on accident, and it lacks versioning and rollback features to recover from these and other situations.

You'll never know who applied a CSS setting or why. If that weren't enough, CSS is something that only designers and some front-end web developers should be managing. There are too many issues that can arise from others changing the CSS of the website.

NOTE

It is highly recommended that you delete all CSS from this setting and/or delete the `Portal.css` file from the file system entirely. Your site will still run just fine if it is missing, and it may even load faster in some scenarios.

Advanced URL Settings

If you are using DNN Platform, your view in this area might look a bit slim. This is because only one feature of this section is exposed to the open source edition of DNN. Those of you using Evoq solutions will likely see an additional section here that allows you to manage how URLs in DNN are put together and responded to. See [Chapters 20](#) and [21](#) for more information about the Evoq solutions.

The remaining section will be consistent across all editions of DNN, which is the Extension URL Providers (see [Figure 4.38](#)). There is nothing to manage here. However, some extensions that you might have built or otherwise installed on your DNN site might include a URL provider for it to help the dynamic URLs it generates to be more human-friendly and SEO-friendly. If you have any URL providers installed, they'll automatically show in this list, and when they do, you'll have an opportunity to enable and disable them as you desire. DNN will not have any URL providers installed by default, so your site will probably have no providers listed.

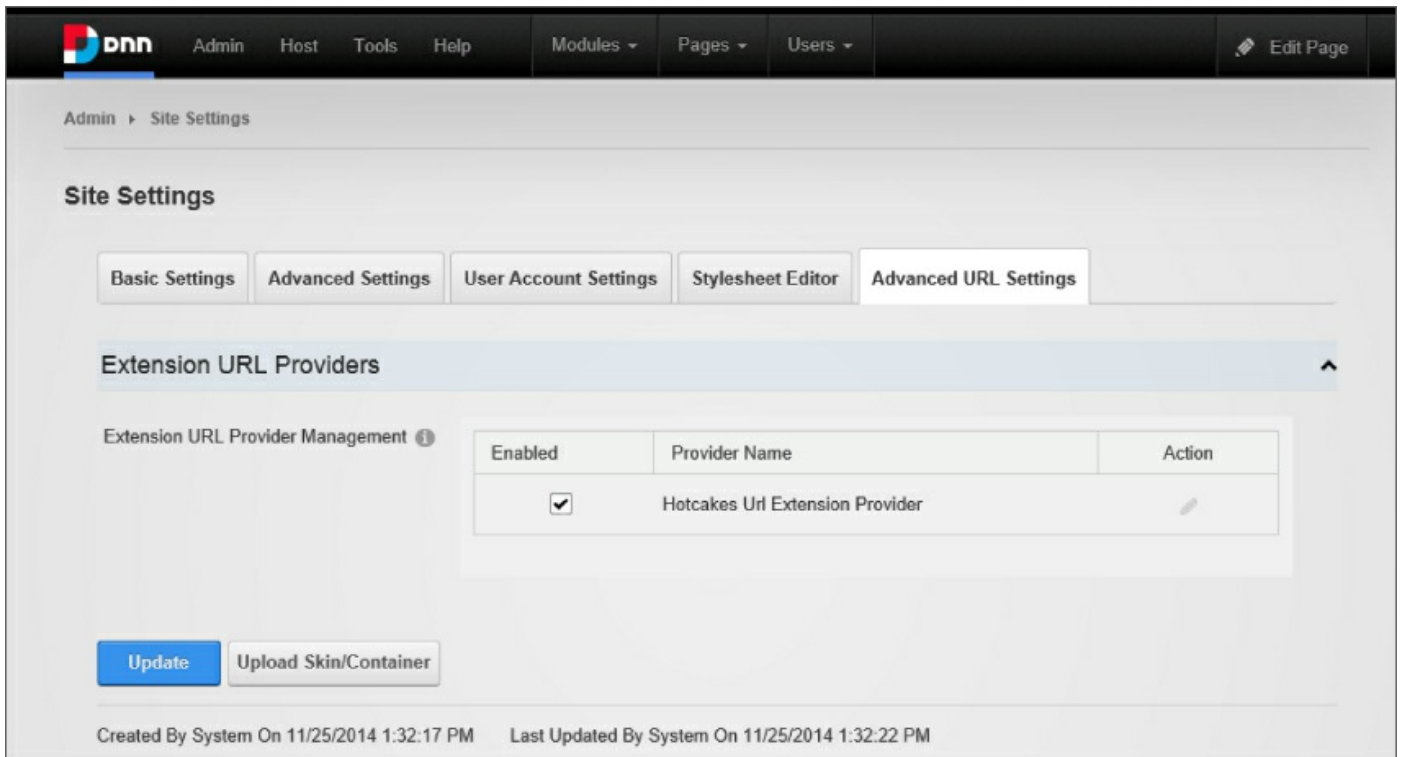


Figure 4.38

Evoq Version of the Site Settings Module

The Site Settings module does not have an equivalent module in the commercial editions of DNN. However, the Advanced URL Settings section does expose more functionality in Evoq Content, Evoq Content Enterprise, and Evoq Suite. You'll find more information about this in [Chapters 20](#) and [21](#).

User Accounts

Part of the power of DNN is having a robust user management system. In fact, in many ways DNN's user management features are unrivaled across the industry without some considerable modifications and/or installation of third-party plugins. If you are using DNN to manage your website users, you have probably selected the right platform for that reason alone.

When you first look at the User Accounts page, you will notice a familiar grid view that lists all of the users in your site (see [Figure 4.39](#)). This grid is deceptively powerful with there being some very useful and lesser known capabilities.

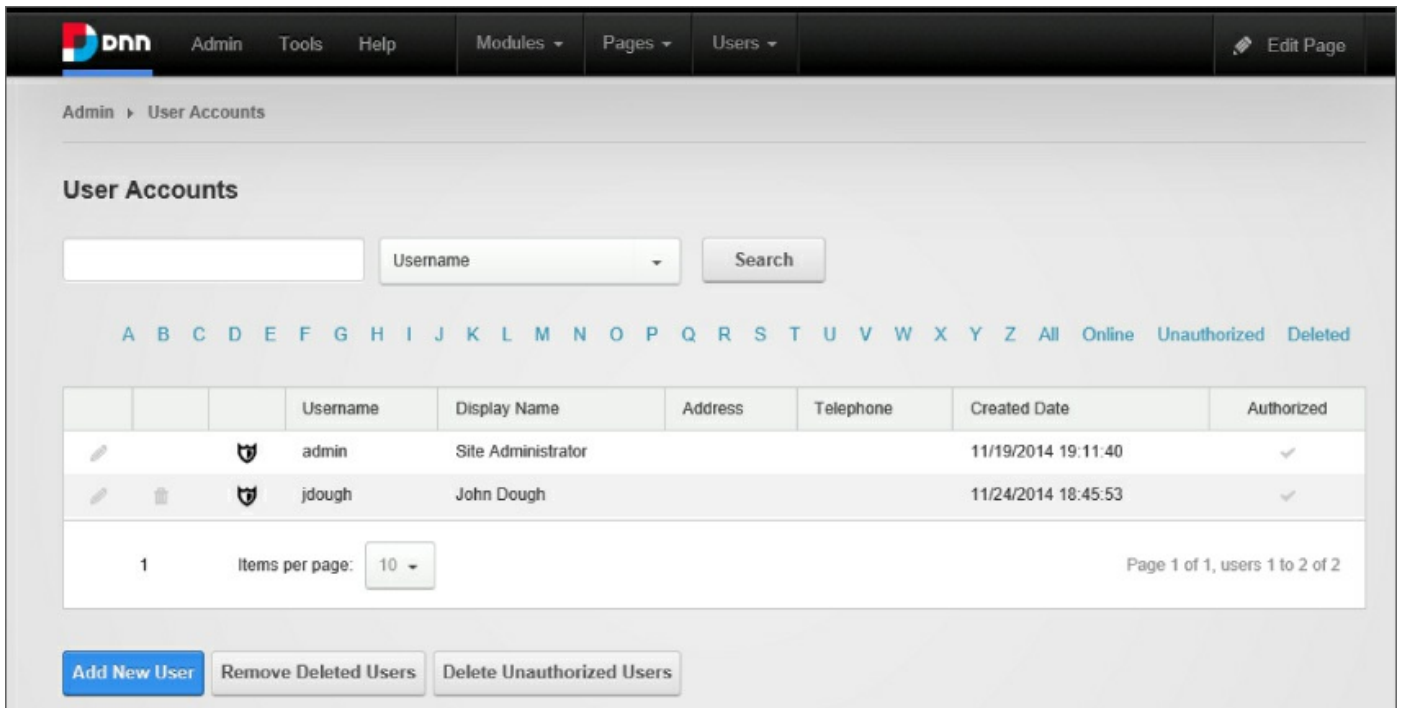


Figure 4.39

You'll be able to discover most of the features pretty easily on your own, so we'll focus solely on some of the key features and tricks that you can use to make your life easier.

First, notice the search textbox. This works just as you might imagine. You can enter a search term, choose how you are searching, and click the button or press the Enter key on your keyboard. You'll be shown the results of your search immediately. But what if you have partial information or want to intentionally search using a partial term? A previously undocumented feature of this search is to use the % symbol as a wildcard. Now, if you want to search for all users of a company, you might choose to search the email field of all profiles for the domain using %@companyname.com. This will return any user accounts that have an email address that ends with @companyname.com.

Next, you'll notice a row of filter options above the grid. They work just like you think that they would. Click the filter and see the results. However, the most useful filters is Unauthorized. Unauthorized allows you to see any user accounts that have been created but not yet verified, or have been marked as unauthorized by a site administrator. This is a very useful filter when looking for users who are having issues activating their account.

The last thing to talk about in this view is one of the buttons on the bottom, Delete Unauthorized Users. This is a great one-click tool to clean up your site.

Over time, the number of both the deleted and unauthorized users will grow. In the case of unauthorized users, it might be growing at a high rate due to spam bots. In just one click, you can delete all unauthorized user accounts.

Deleting users using this button or the other delete action provided will not permanently delete the user, but rather “soft delete” the user. This keeps the user account in the database for referential integrity but removes all other capabilities from the user. Once deleted, you'll have the option to either permanently delete the user or restore the user if the deletion was a mistake. In some situations, it may be a legal requirement to either always soft delete users or vice versa. You should be careful to know this requirement for the specific site you're responsible for prior to permanently deleting users.

User Account Settings

It would be useful to know how you can configure how the Security Roles module works before we discuss using them. The Security Roles module has settings unique to it, like many modules do. [Figure 4.40](#) shows the User Account Settings view. It is the fourth tab after choosing Module Settings.



NOTE

This module has had a few different names over the years. As a result, you'll see this module referred to as any one of these names in various community resources and documents and even in the various administrative views. The actual name of this module in the core of DNN is Users and Roles. You'll notice that in the Control Panel, it's referred to as Manage Users and User Accounts.

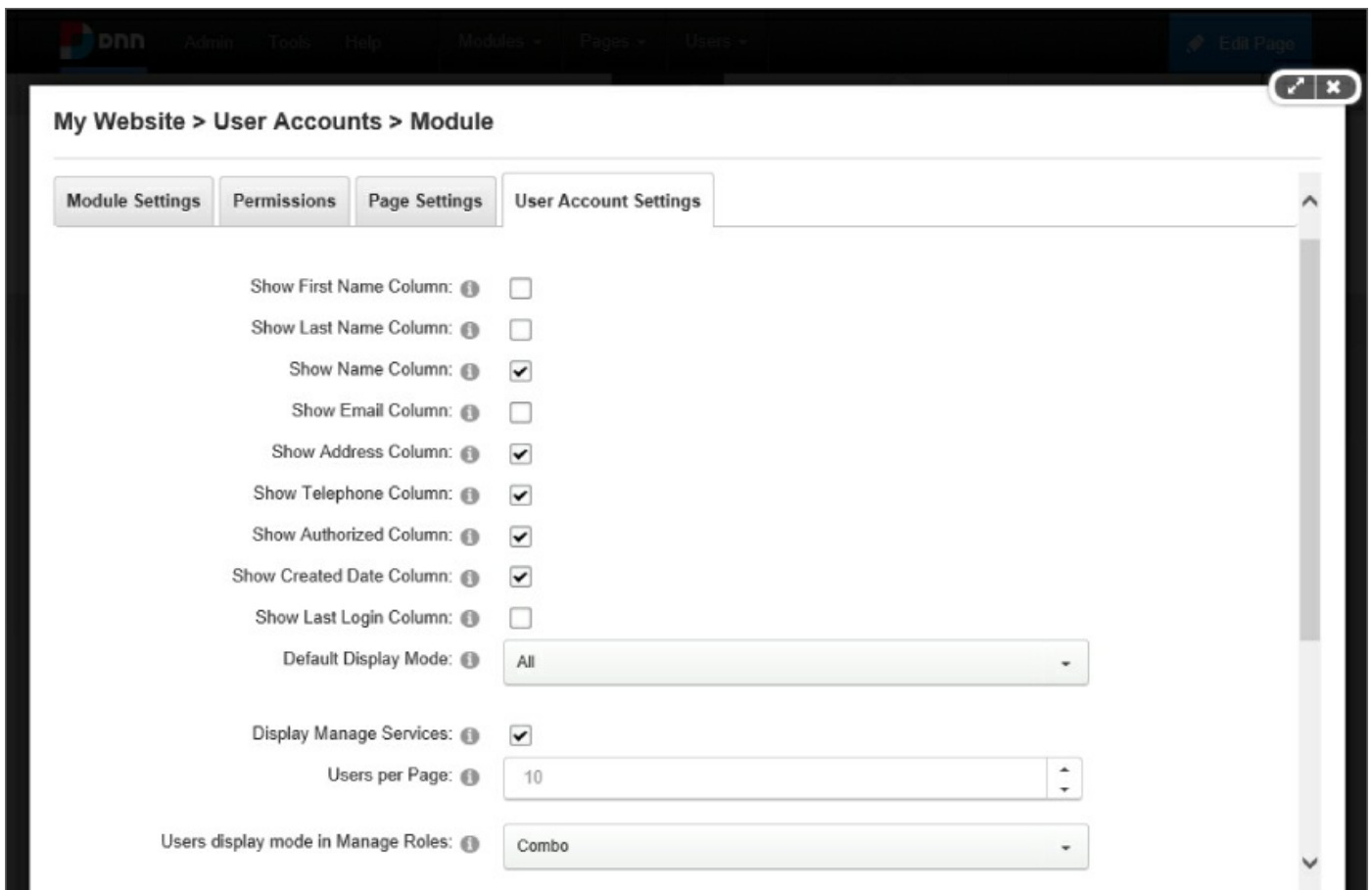


Figure 4.40

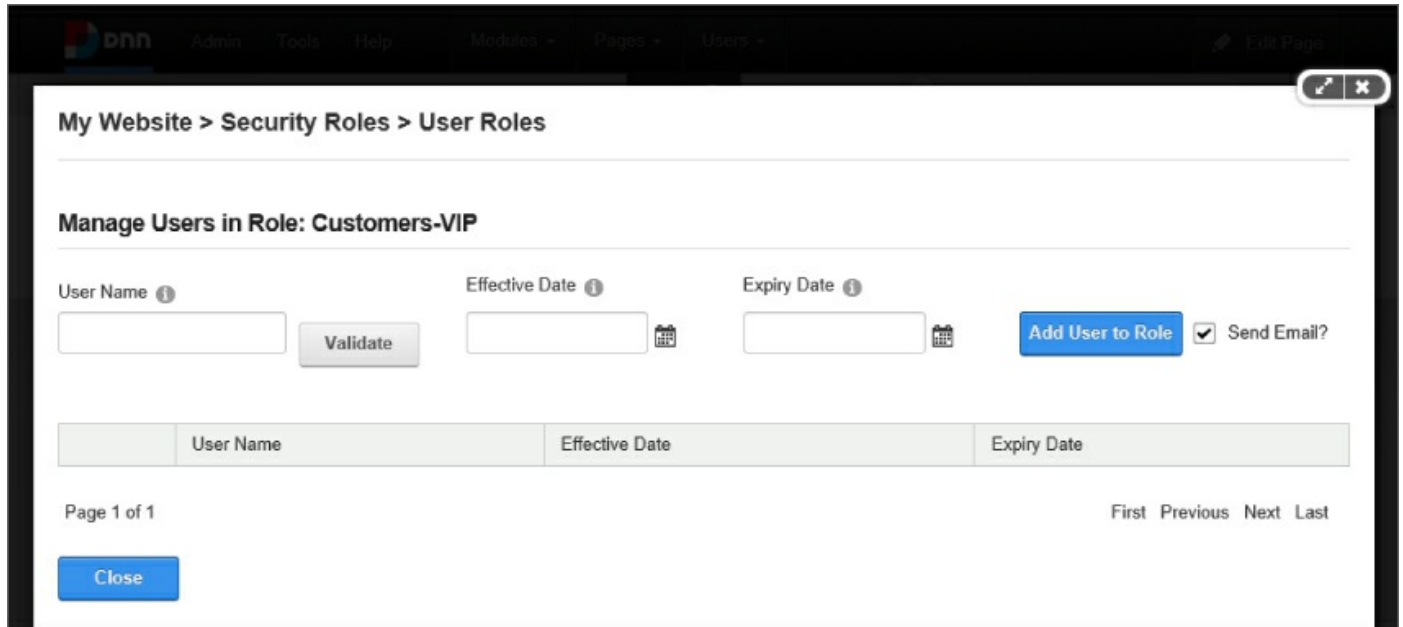
User Account Settings allows you to alter how the module displays the columns and a few other features. You can check and uncheck the various columns that you would like to have shown in the default view. For example, you may not want to have the telephone or address columns shown in the users grid, even though this is a default setting. You might want to show the email address, though. This is a checkbox away for you.

The next useful feature to explain is the Default Display Mode. This drop-

down list allows you to determine how the users grid is shown on the first page load. Do you want to list all users in a paged format, or do you want to display none and rely on the search and filters? Most sites tend to use the latter.

Display Manage Services may not make much sense to you at first. That makes sense because it doesn't really have anything to do with this module but instead is a setting to show or hide the ability for members of your site to see and manage any public roles that they're a part of. Unchecking this setting will remove this capability from their profile.

The final setting we'll discuss here is the User Display Mode in Manage Roles setting. This setting will default to providing a drop-down list populated with users in the respective view. When a site first goes live, this is not a problem, since there are not that many users to scroll through. However, once you begin to approach 100 or more users, the drop-down list is no longer usable. In this scenario, you would want to switch the setting to TextBox. Doing so will replace the drop-down list with a textbox and a validation button. Now, you'll be able to enter the username and click a button to verify that you entered a correct user, as shown in [Figure 4.41](#).



[Figure 4.41](#)

Evoq Version of the User Accounts Module

The User Accounts module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for

commercial editions.

Advanced Settings

As discussed earlier, the Admin menu of the Control Panel splits the various settings and views into two categories. You just learned about the common settings, so now you can learn more about the Advanced Settings section. Advanced Settings is a deceiving label, though. There's nothing advanced about these settings at all. They're simply uncommon settings. Don't let the label deter you from exploring the various views in this section of the Admin menu. There are a lot of useful features here.

The Advanced Settings sections contain a lot of features that are infrequently used. They either are set-it-and-forget type of things or require management maybe as often as once a year. As a result, you'll notice two things. First, we will not discuss all of these settings in detail. Second, some of these settings/views are likely to be removed from the platform in the near future and placed into the open source DNN Forge (<http://www.dnnsoftware.com/forge>). This allows those features to be more easily maintained by the community, while still allowing them to be used on sites that need them without adversely impacting the sites that don't. Another side effect of this will be that the Admin menu will be cleaner and more streamlined.

Advanced Configuration Settings

This is still a relatively new feature in the life span of DNN. If you've been around more than a few years, you'll recognize what this feature is immediately. If you haven't, you should know that this feature contains the common setup steps that used to be part of the installation wizard that runs when you install DNN. In the interest of usability during the installation process, these steps were removed from the wizard and placed in this module. This allows administrators to quickly make these adjustments to their sites, but after installation has already occurred.

NOTE

If you are logged in as a host/superuser, you will see six tabs of settings to manage. As an administrator, you will see only one tab because the other features require a higher level of permissions.

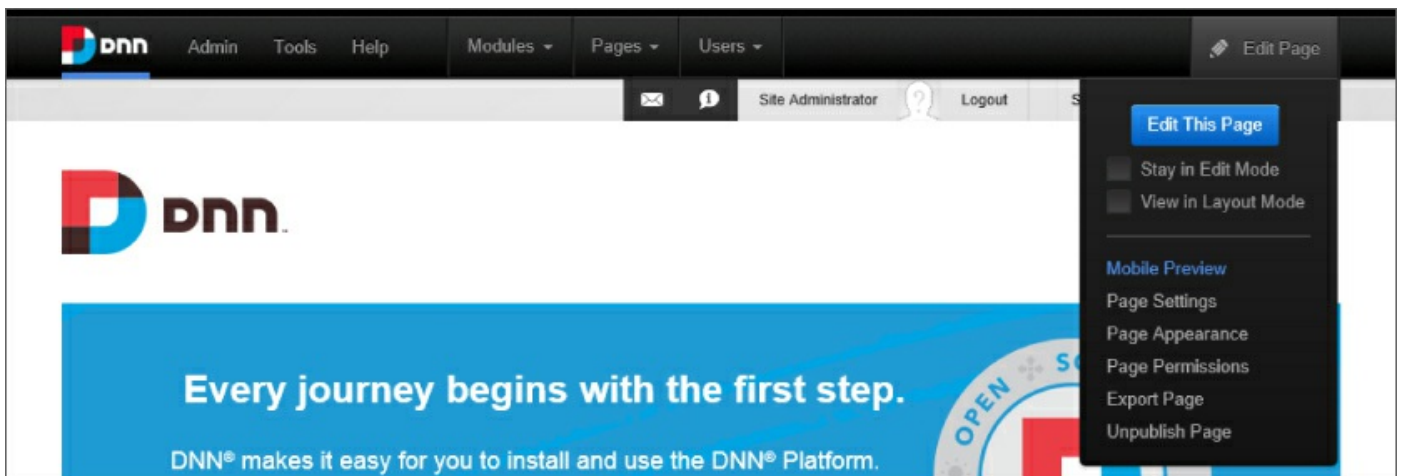
While this feature is very convenient because it places a large number of common configuration options in one place, these configuration options are all available in their respective setting views as well. It's probably a good idea to manage these settings there and not in this module. It is likely that this module will be removed in a future release.

Evoq Version of the Advanced Configuration Settings Module

The Advanced Configuration Settings module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Device Preview Management

You may not be aware of this, but DNN has a very useful Mobile Preview feature in the Control Panel that allows you to view pages on your site as a mobile visitor might see them. You can choose the device you want to preview as, switch to landscape mode, and more. [Figure 4.42](#) shows where this feature can be found.

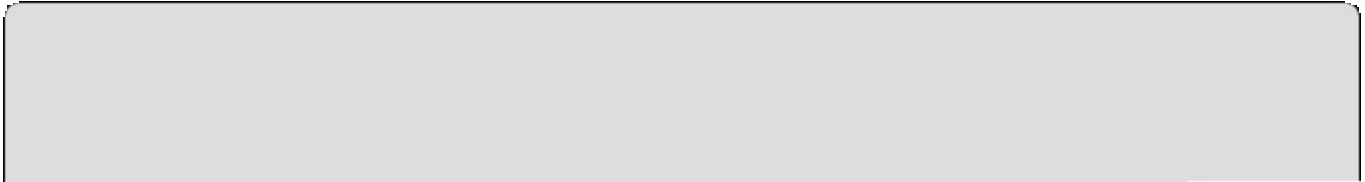


[Figure 4.42](#)

The Device Preview Management page manages this feature. When you arrive

at this page, you'll be able to see, manage, and add devices to preview pages as.

While this sounds like a lot to manage, you don't have much to worry about. The back-end of this feature contains data that is managed and imported regularly for you. The device data is automatically updated on a quarterly basis from 51Degrees (<http://51degrees.com>). However, new products will not show in the list of default devices automatically. You'll need to add them yourself, but the data will be there.



NOTE

If you want the data to be updated more often, 51Degrees sells a subscription to their data in the DNN Store. Just search for 51Degrees, and you'll find it. See <http://store.dnnsoftware.com>.

Evoq Version of the Device Preview Management Module

The Device Preview Management module does not have an equivalent module in the commercial editions of DNN. In fact, this module used to be in Evoq Content only but was later added to all DNN editions.

The commercial versions of DNN will update your device data on a weekly basis, versus a quarterly basis as discussed previously. If you are an Evoq customer, you don't have to purchase the 51Degrees subscription.

Extensions

If you have host/superuser access, you might already be confused about there being an Extensions page in the Admin menu. This is because this Extensions page has a different purpose and exposes different features than what is in the Host menu. This instance of the Extensions page allows you to manage who is allowed to add the individual modules to pages on the site. By default, only administrators can add modules to pages.

If you want others on your site to be allowed to add specific modules to a page, you'll first need to have those users in a security role and then apply that security role to a page so that it has page-level edit permissions. This is labeled either Edit Page or Full Control in the page permissions, depending on the edition of DNN you're using. Everything else about this feature should be self-explanatory.

Your site might have additional authentication providers installed. An authentication provider is simply an additional way for a user to log in on your site. Your site might have only one or many authentication providers. Another use of this page is to enable, disable, and configure any installed authentication providers, since they generally work on a site-level basis.

All other extensions and extension types are listed here too, but you'll only get a read-only view of who made them and how to get support—if the vendor

provided that information.

Evoq Version of the Extensions Module

The Extensions module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Google Analytics

Google Analytics is probably the most popular way to add and manage the statistics about traffic coming to, spending time on, and leaving your site. It's for this reason that this third-party service is a built-in feature in DNN. You can easily copy and paste your Google Analytics ID into the given setting and automatically have the required scripts added to all pages. This feature is incredibly simple and generally only needs to be set once for the duration of a website.

Evoq Version of the Google Analytics Module

In the Evoq solutions, you'll find this page labeled as Google Analytics Pro instead of simply Google Analytics. You have several additional features that may be of use to you, including advanced parameters and segmentation rules to track groups of people differently. If you don't know what those features are, chances are that you won't use or need these features anyway.

Languages

DNN has pretty much always been a great platform if you want to support more than one language on your website. There has been a strong focus on the multilingual features for a very long time. While English is of course the primary supported language, German, Dutch, French, Italian, and Spanish are all officially supported languages on every release. That being said, you technically can support any other language that you want as long as you have or create the appropriate translations.

On most sites, you'll first install DNN using the primary or fallback language. This is the language that you'll be looking at the majority of the time. Once you install one or more additional languages, you'll see them listed on the Languages view. When you look at each language, you'll notice that you have some editing capabilities (see [Figure 4.43](#)).

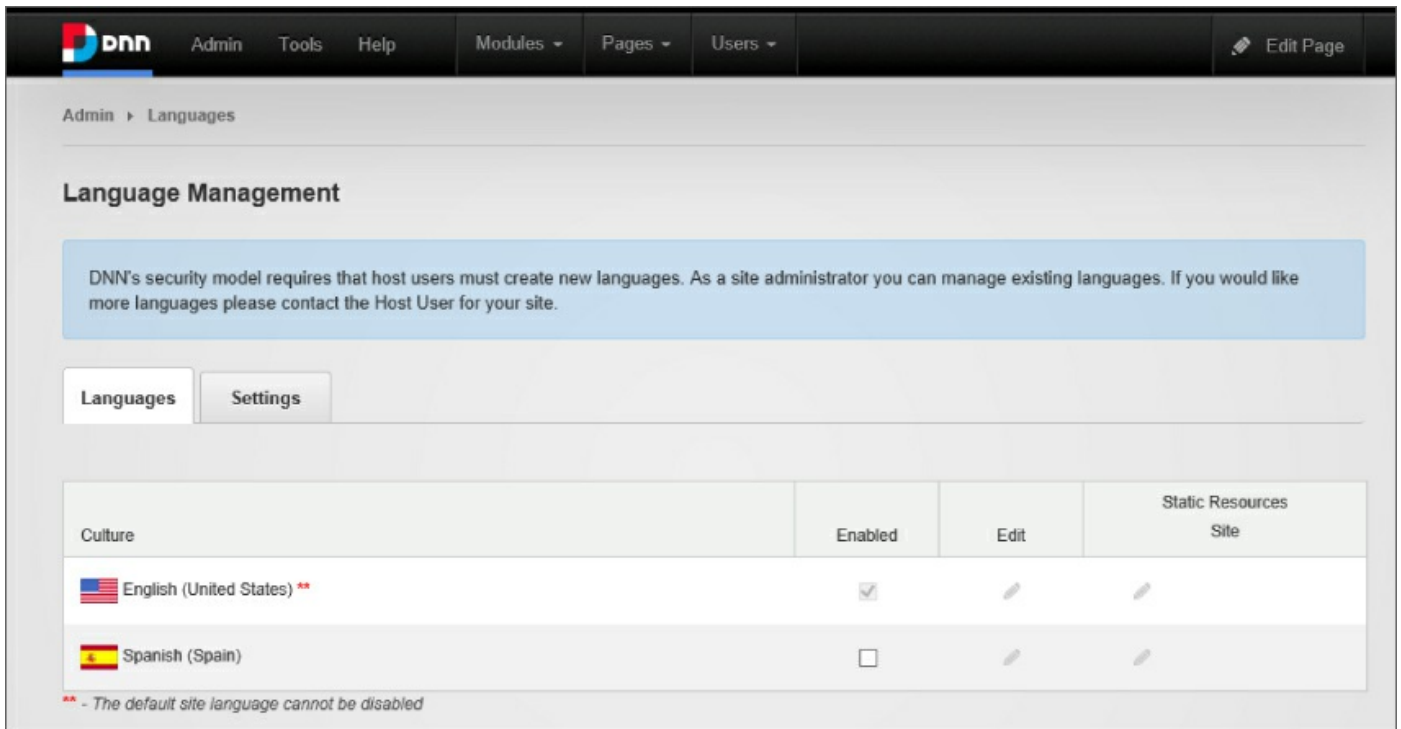


Figure 4.43

First, you can see whether the language is enabled. Installing a language alone does not make it available for use. You need to enable it. Next, you can also edit the language, but the only real use for this is when dealing with additional languages. You will want to make another language the fallback language. This is useful when language translation is not available, because the fallback language will be used.

Next, notice the editing features for Static Resources, where you can edit System, Host, and Site. These editing areas allow you to browse through and manage the translations for that language on the level that you choose. You should be careful about which level you choose when adding or editing translations.

System allows you to edit the default language files for the installation of DNN. Host and Site allow you to override the System translations for either the host or site level. If you manage translations at the host level, those changes will be reflected on each site if you have multiple sites on your instance of DNN. Editing the site language files will affect only the site that you're on. The other sites will continue to use either their site- or host-level translation.

At the bottom of this module, you'll see several buttons that allow you add and manage languages in other ways. Most of these are outside of the scope

of this chapter, but you can learn more about them in [Chapter 10](#). However, the most useful button to mention right now is the Install Available Languages button. Clicking this button will take you directly to the Extensions page, where you can install one of the previously mentioned languages that ship with DNN. Within a few clicks, you'll be able to install and enable one of these other common languages.

Evoq Version of the Languages Module

The Languages module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Lists

Out of the box, this feature is used for two things that a site administrator will be concerned about, which are to manage the banned passwords and the profanity list. When you ban passwords, they should be put into this list, and the same for any profanity words that you want to include in DNN's profanity filters. You'll find that managing these lists is incredibly simple.

You can create your own lists as well, which comes in handy for features that require a reusable and manageable list of information. While you can do this yourself and you'll be able to manage and create lists, this feature is mostly populated by developers through their modules. For example, some third-party modules might store their categories and other information in this module. Now you know where to go to manage their lists.

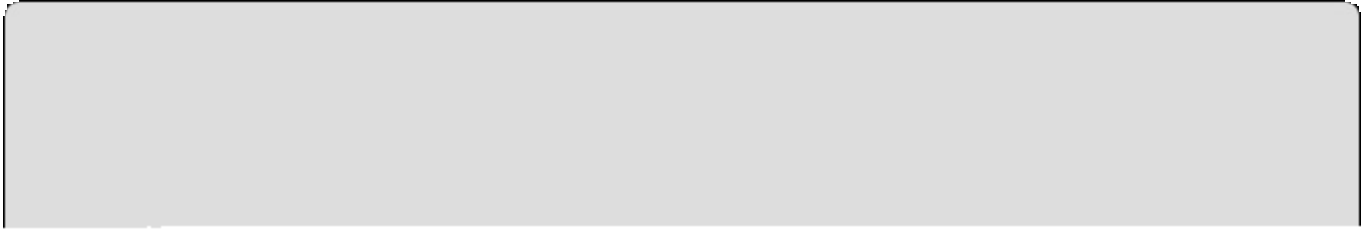
Evoq Version of the Lists Module

The Lists module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Newsletters

The Newsletters module provides a very simplistic way to generate emails to members of your website. You can easily create a plain-text or HTML email for members of your site and even include an attachment. However, it lacks many of the features that even a low-cost email marketing solution offers, such as link tracking, campaigns, templates, and reports. For these reasons and more, you should consider using a third-party solution to generate your

email campaigns.



NOTE

This feature is one of those that will likely be removed from the core of DNN and placed into DNN Forge in the future.

Evoq Version of the Newsletters Module

The Newsletters module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Search Admin

It wasn't until recent versions of DNN that this feature became useful again. Since version 7.1.0, the search engine has been replaced with a much more functional alternative that's based on the very popular [Lucene.Net](#) search engine. It unleashes a lot of power and capabilities that were not before available to all editions of DNN.

NOTE

[Lucene.Net](#) is a very popular search engine that is used on a large number of websites worldwide and provides a world-class onsite search experience. [Chapter 11](#) will discuss this in more detail.

The first thing that you can do in this module is reindex the site. As the module shows you, this feature deletes the existing site index, and a new index will be created the next time the site is indexed by the search crawler. When that happens depends on your Scheduler settings in the Host menu. This sounds great on the surface, but you should know that doing so could create a potential performance impact on your site, making it unusable to your visitors. This of course depends on the size of your site and how much traffic you have.

You can add synonyms to your site, which allow the search to be more flexible for members of your site. The example of a synonym that's included is a great one. It has DNN and DotNetNuke listed. If anyone searches for either term, the search engine knows that the term is interchangeable and will index those terms appropriately. Another example, for a recipe site this time, is “hotcakes” and “pancakes.” Essentially, the person searching for either of those terms gets similar results.

“Ignore words” are also known as “stop words” in the technical community. These are common words that should not be used as part of a search because the results would be too broad and not at all useful. You should feel free to add your own if your industry has words that work the same way, but it's not a good idea to delete any that are in the default list. This is a common practice that's used by even the major search engines, such as Google, Bing, and Yahoo.

Evoq Version of the Search Admin Module

In Evoq Content, Evoq Content Enterprise, and Evoq Suite, you'll find that this feature also includes a section for Advanced Crawlers. This feature allows you to manage some of the more advanced features such as duplicate results, additional web properties to crawl and index, included/excluded file extensions, and more. The Evoq solutions also come with additional crawlers. You can find out more about this topic in [Chapter 20](#).

Search Engine Site Map

A very nice feature of DNN is that it ships with an XML sitemap. The reason that this is so nice is that search engine tools such as Google Webmaster Tools use XML sitemaps to help manage the way a site is indexed. From the moment you install DNN, you not only have an XML sitemap, but your sitemap is constantly updated for you. That being said, you do have some configuration options available to you to help you manage how your sitemap is generated.

While there are a handful of settings here to possibly configure, the main thing is how much you want to manage your XML sitemap on your pages and for how long those settings should be cached. The larger your site, the more important it will be for your SEO strategy to enable Page Level Based Priorities. This allows you to define the importance of a page in the respective page settings, which will then be reflected in the resulting XML sitemap.

There aren't any wrong ways to set up this feature as long as you plan and measure your impact regularly.

Evoq Version of the Search Engine Site Map Module

The Search Engine Site Map module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Site Log

This page is a legacy feature and should be turned off on every site. While Google Analytics (Pro) is a great replacement for this feature, using *any* other analytics service other than the Site Log is highly recommended.

The Site Log does not include the granular detailed information, reports, or other features that you might be looking for in an onsite analytics tool. Also, enabling this feature can cause a significant adverse impact on the overall performance of your site.

NOTE

This feature is one of those that will likely be removed from the core of DNN and placed into DNN Forge in the future.

Evoq Version of the Site Log Module

The Site Log module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Site Redirection Management

Site Redirection Management is another companion module to the previously discussed Device Preview Management module and the Mobile Preview feature. This specific module allows you to manage how mobile visitors are targeted and/or redirected when they arrive on your site.

Every site owner has a decision to make, when they decide that they want to cater to their mobile traffic. Are you going to use a technique like responsive design, or are you going to have a mobile-specific site? There are plenty of reasons to choose either option, depending on the type of site you are running and the type of visitors you're catering to. Debating those pros and cons is outside of the scope of this book, but do know that you can do both—responsive/adaptive design and a mobile-specific website? This feature will aid you in doing the latter.

When first entering this module, you're given two choices. Do you want to create a simple mobile site redirection rule, or do you want to create one from scratch and define exactly what and how you want your mobile traffic redirected? In either case, you can end up creating similar redirection rules for a multitude of devices. If you want to create a single rule to redirect all mobile traffic to another version of the site, such as m.yourdomain.com, you can do that. Also, if you had different sites that each target and render content to Windows Phone, Android, and iPhone differently, you can do that too—and more.

Evoq Version of the Site Redirection Management Module

The Site Redirection Management module does not have an equivalent

module in the commercial editions of DNN. However, it does have a larger subset of data. The data that is included from 51Degrees on DNN Platform has only 51 of the 125 different properties that you might want to target on various devices. One of those properties is the ability to target a tablet differently than mobile and desktop visitors. To learn more about your options with 51Degrees, refer to the section earlier in this chapter called “Device Preview Management.”

Site Wizard

The Site Wizard is one of those features that you might use only once throughout the entire lifetime of your website. It allows you to use a site template (discussed in [Chapter 5](#)) to create a new website—at least that's the intention. In short, this module will use a site template as part of an overall wizard process to select default skins, containers, and more to create your website.



TIP

If you're going to use this feature, it is highly suggested that you first play with this feature on a local instance of DNN that you can delete and re-create. It is very easy to completely delete a site if you're not familiar with how this feature works.

If you're careful, you can also use this feature to create an entirely new section of your website. How well it works depends on the quality of the extensions you have installed on your site, because some will move content and settings better than others. Prior to using this module, you should ensure that all the extensions in the site template are installed and upgraded to the same versions in the new site. Also, any modules that do not support an API technology known as `IPortable` will not be able to import content. This is something to note if content is part of the site template.



NOTE

This feature is one of those that will likely be removed from the core of DNN and placed into DNN Forge in the future.

Evoq Version of the Site Wizard Module

The Site Wizard module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Skins

The Skins page is not used that often. Generally, it's used during the development of a new site and sometimes is used during the rollout of a new site brand. It allows you to view the available skins and containers that have been installed in your site and then apply them as the default for new and existing pages and modules. You might remember a similar feature from the Site Settings discussed earlier. You can use that too, but the Skins page is much more visual and results in a much more usable experience.

There are a couple of notable differences with the settings found in Site Settings. When looking through the various skins and containers that might be available, you'll be shown thumbnails of what the skin or container might look like on the page (in addition to the preview button).

The other difference is that you can override the settings specified in the skin objects that are in the skin or container. For example, if the TITLE skin object is applying a CSS class that you don't like, you can use the feature shown in [Figure 4.44](#) to apply a new class name of your choice. You should use this feature carefully, as the changes you make here will potentially impact other sites in your DNN instance. You'll find this at the bottom of the page, but its features are available only to host/superusers.

Edit Skin Attributes

Skin ⓘ <Not Specified>

Container ⓘ Gravity

File ⓘ Title_h2

Token ⓘ TITLE

Setting ⓘ CssClass

Value ⓘ MyCustomClass

Update

Sets the CSS class used to render this skin object

Figure 4.44

Evoq Version of the Skins Module

The Skins module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Taxonomy

The Taxonomy module has been very creatively used by people in the DNN ecosystem for more than just tagging. Those who think creatively have used this feature to create some amazing solutions, including one by DNN Corp. co-founder Nik Kalyani, who used this feature to create an open source mobile solution that content editors can use, called Druid (<http://druid.hypercrunch.com>). Anyhow, such uses are more advanced and are outside how you might be using this feature.

The Taxonomy module allows you to create one or many vocabularies that can then be used for any number of purposes. The most common purposes include tags for social features, and you'll also be able to manage the module categories that appear in the Control Panel from this feature. If you're working with multiple modules that have tagging features, chances are that each of them is storing its tags here.

Evoq Version of the Taxonomy Module

The Taxonomy module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Vendors

The Vendors module has always done a great job of providing banner advertising of various kinds on your DNN site, but it has always been lacking in usability and improvements over the years. That, coupled with third-party or offsite advertising solutions, has ensured that this feature hasn't been as highly adopted in recent years. That being said, it is the easiest way to manage onsite advertising and value propositions if you're not using an offsite service like DoubleClick.

If you want to serve your own banners and other ads from affiliate programs like Amazon, you only need to add a vendor for each entity that will have its own banners. Then, you'll be able to add as many banners as you would like for each vendor of various types. When you add banners, you have the ability to define groupings for them. When used this way, it's easier to display advertisements of similar contexts and sizes using the Banners module and Banner skin object.



NOTE

This feature is one of those that will likely be removed from the core of DNN and placed into DNN Forge in the future.

Evoq Version of the Vendors Module

The Vendors module does not have an equivalent module in the commercial editions of DNN, nor does it have any features reserved for commercial editions.

Best Practices for Site Administrators

This section is meant to give you some insight into how other people run their DNN sites using techniques that have been developed over time on production sites, both small and large. It should be noted that although this section is clearly labeled in a way that best practices will be discussed here, numerous best practices have been discussed in previous sections that site administrators should learn and implement. Also, as your site needs and complexities grow, you may find that you'll need to make adjustments to how you adopt best practices such as these.

Getting User Registration Right

As a rule of thumb, keep registration set to None unless you absolutely need to allow visitors to create accounts on your website. The moment you open registration to anonymous visitors, you are entering an entirely new level of issues. The most obvious issue is the creating of spam accounts. There are an unknown number of bots out there that specifically target DNN sites to create new user accounts that contain spam content. The more popular your site, the more accounts will be created. Some sites have reported to have as many as three spam accounts created per second, but this is an extreme example.

You shouldn't be afraid of allowing visitors to create new accounts. However, if you do allow this, consider the following options for the best user experience for all parties:

- Block requests in Internet Information Services (IIS) 7 for the path `ctl=Register`.
- Block requests to the default registration page and create your own.
- Use a third-party registration module instead of the default.
- Add a new required profile property to your registration form.
- Regularly delete all unauthorized user accounts.

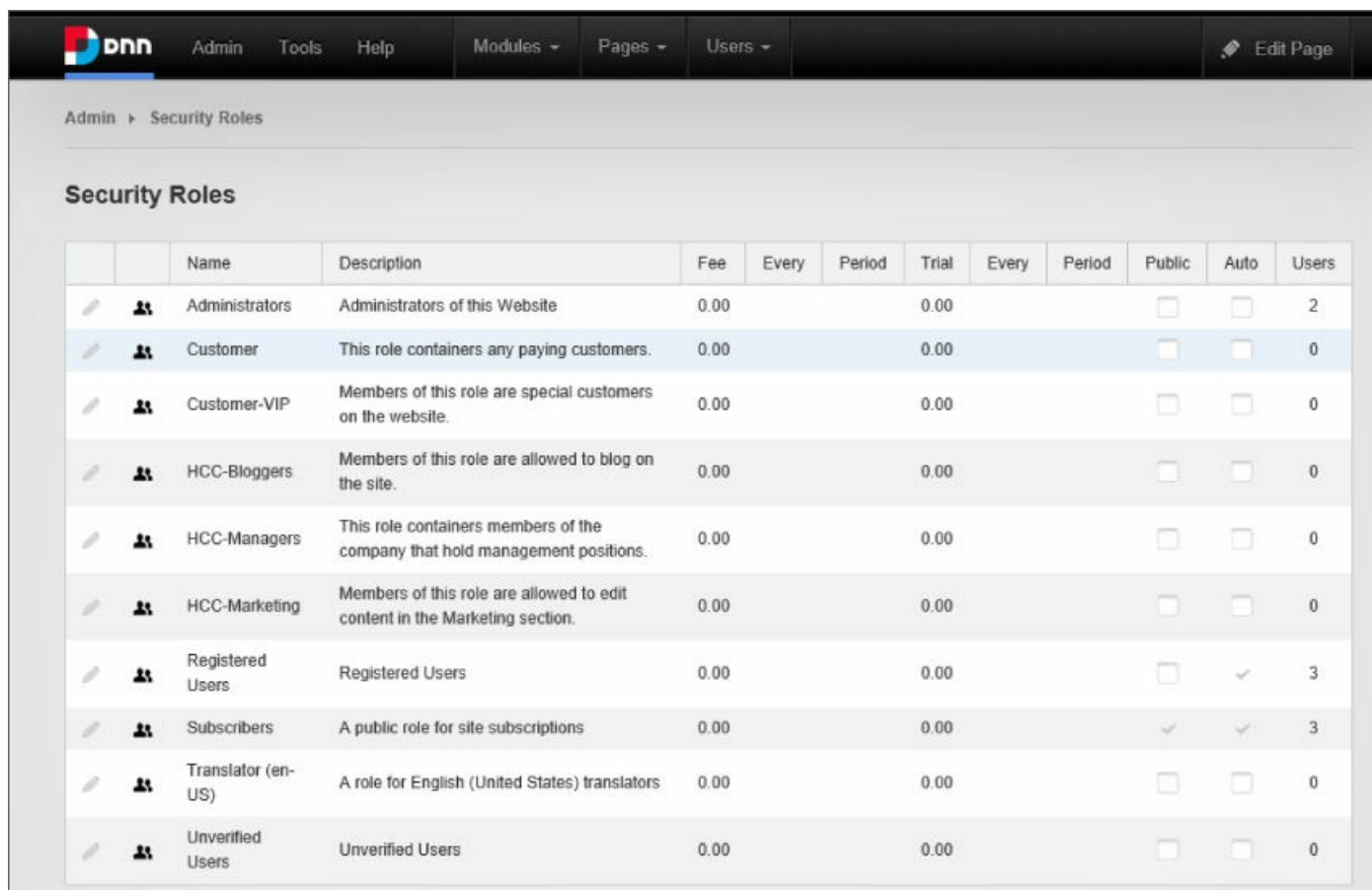
Having a Naming Convention

There are many repetitive tasks in DNN that require labels, such as security roles, module titles, and more. Having a naming convention saves time in creating and managing the various objects you're dealing with. It also makes it easier to define and enforce process for others that may be contributing to

the site management. Overall, a naming convention policy will make everyone's lives easier.

First, objects should always have a title. If they don't have a title, there are numerous usability issues that may be created as a result, not the least of which is how to handle items in the Recycle Bin. If you don't want to display the title, nearly every feature has a way for you to hide it.

Outside of that, everything in your environment should have a common naming convention where it makes sense. A prime example of this for site administrators is security roles. You should consider a naming convention that prefixes all similar objects. The Security Roles section earlier in this chapter gives you a great example of this. [Figure 4.45](#) shows how clean this can look.



	Name	Description	Fee	Every	Period	Trial	Every	Period	Public	Auto	Users
	Administrators	Administrators of this Website	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	2
	Customer	This role containers any paying customers.	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	Customer-VIP	Members of this role are special customers on the website.	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	HCC-Bloggers	Members of this role are allowed to blog on the site.	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	HCC-Managers	This role containers members of the company that hold management positions.	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	HCC-Marketing	Members of this role are allowed to edit content in the Marketing section.	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	Registered Users	Registered Users	0.00			0.00			<input type="checkbox"/>	<input checked="" type="checkbox"/>	3
	Subscribers	A public role for site subscriptions	0.00			0.00			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	3
	Translator (en-US)	A role for English (United States) translators	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0
	Unverified Users	Unverified Users	0.00			0.00			<input type="checkbox"/>	<input type="checkbox"/>	0

Figure 4.45

Never Sharing User Accounts

This is something that is unfortunately an incredibly common thing to come across. People who maintain content or configuration responsibilities may be

sharing the same login credentials. There might be several or an entire company of people using the same admin or host/superuser account. This is not only bad practice, but it's much more trouble than it's worth.

What if you have to let someone go or they quit? What if someone writes down their credentials somewhere? What happens when someone finally changes it and doesn't tell anyone or something happens to them? What if someone changes a configuration option that they weren't supposed to touch? How do you apply accountability?

Yes, there are ways to deal with these and other similar problems, but you can prevent them altogether by never sharing user accounts.

Everyone with content editing, administration, or host/superuser permissions should not share their user account. Every user account should be unique to a specific person. This seemingly small rule can prevent a multitude of troubling issues.

Editing as a Content Editor

If you speak to security experts in any technology, one of the most common things they'll tell you is to lower the surface area of attack. This means reduce the number of ways that someone can potentially penetrate your defenses. Imagine being an archer and being told to hit the target while it's turned sideways. It's a similar principle.

You should never perform content-editing tasks when you're logged in as a privileged user, even on closed networks. Security vulnerabilities and other exploits have been found in even the most secure networks.

If you are charged with the responsibility for maintaining the site on an administrator or host/superuser level as well as content editing, you should have a second user account that does not have your higher-level privileges. Not only is this more secure since your day-to-day account will not have the higher privileges, but it will also prevent you from circumventing processes just because you can.

Fewer Administrators, Not More

Whether you're talking about site administrators or host/superusers, you should never have more than you need. The fewer people you give the ability to manage the site's configuration, the better off the overall health of your

website will be over time. Never promote someone to site administrator or higher simply to save time. This little shortcut will end up giving you many more headaches later.

In general, most sites have up to three host/superusers and about the same number of site administrators. Even though you can create as many as you want of each, three is a good rule of thumb. This is certainly one of those areas where the phrase “less is more” truly shines. The fewer the number of privileged users, the lower your long-term support costs will be.

Fewer Permissions, Not More

Something that happens more often than anyone would like to admit is that they promote a poweruser to administrator because they need to create a lot of content that includes pages and dropping multiple modules on the site. It's far more simple and faster to make them an administrator than to create permissions and content for them to manage. This is true, at least initially.

No matter how innocent a poweruser's intentions, eventually this scenario leads to trouble. Even though they were told not to do so, they'll slowly begin to administrate configuration and content in areas of the site that they shouldn't touch. Sooner or later, you'll need to revoke their administrator access and do what you were supposed to do previously, but at this point the damage is already done.

When you need to give a poweruser more privileges, you need to do the following:

- Define the scope of their planned changes.
- Create a new security role for them (if necessary) and add them to it.
- Create the pages for them.
- Assign them page-level edit permissions (only if absolutely necessary).
- Add the necessary modules to the new pages.
- If you didn't give them page-level permissions, give them permissions to edit the necessary modules on the page.

Yes, this process will take a bit of time on your part, but such a thing is necessary to ensure that your site lives a long and trouble-free life. The less someone *can* do to your site, the less they *will* do to your site.

Never Using “User” Permissions

User Permissions is another shortcut method that seems convenient but is much more trouble than it's worth in practice (see [Figure 4.46](#)). It's far too easy to assign permissions to a person than to do the extra step of creating a security role for them. If you've done this even once, you should instead consider creating a security role for them, even if you don't anticipate ever adding anyone to that security role in the future.

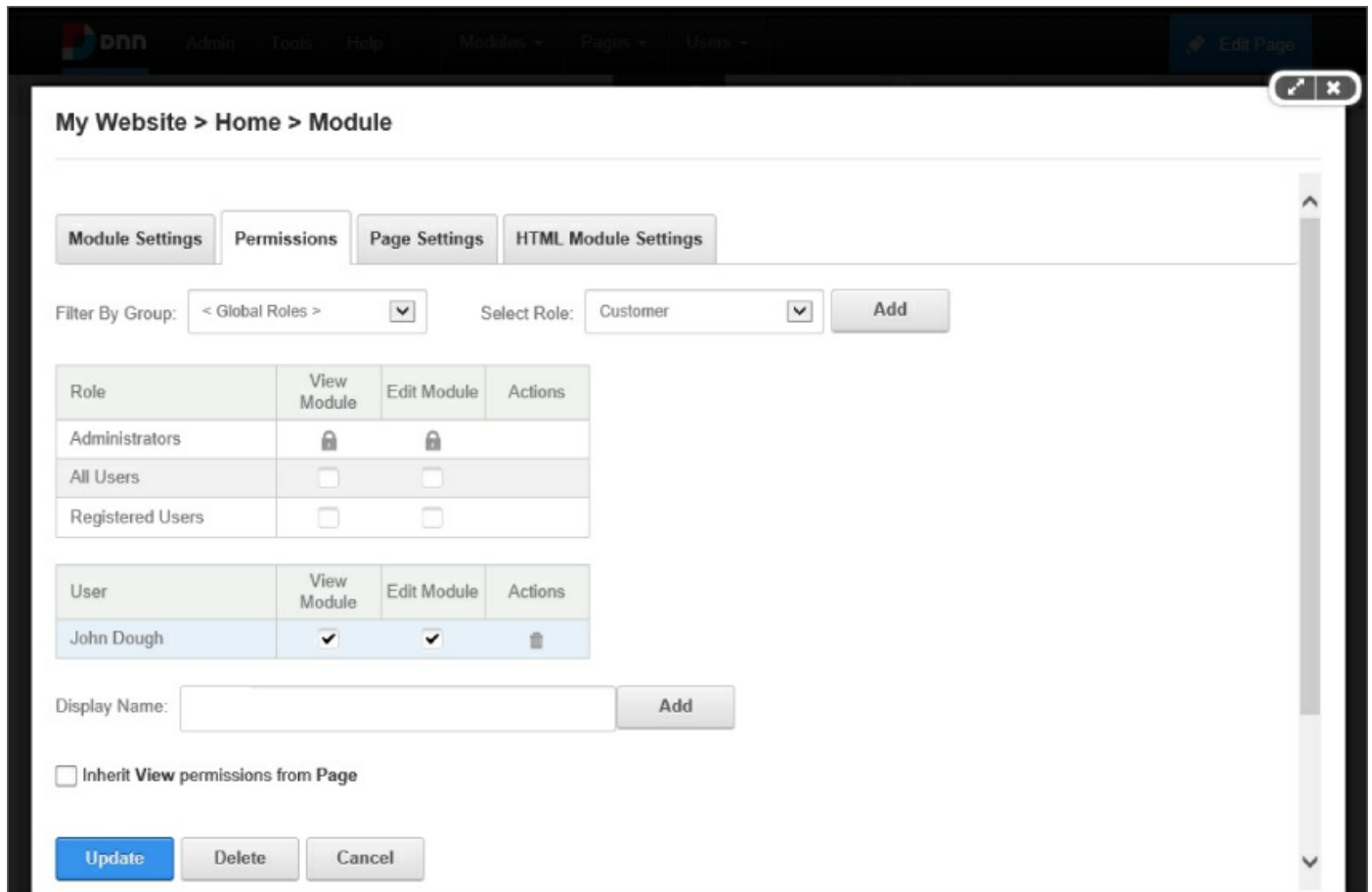


Figure 4.46

What happens if that person quits, is promoted, or is asked to leave their job? How do you re-apply their security settings to the next person who replaces them? How do you revoke the permissions you've already applied? Simply put, in most cases you can't—not without getting someone more technical involved. How much does that cost the company versus saving the five extra minutes it would have taken to do it right the first time?

Permissions audits are already a complicated thing to do. Adding a user to the permissions grid directly instead of using security roles only further complicates this process.

Basically, whenever you need to allow a person to do something that they're not already doing, use a security role, every time. You never know when you need to replace someone, add someone, or extend their permissions to other areas.

Having a Process and Standards

If you've followed any of the previous advice and instructions on how to administrate your website, you are probably already thinking about this topic. You need to have well-defined processes and standards. The process and standards need to be universally distributed to all site contributors and enforced judiciously and equally across all stakeholders. Doing anything else will only create more of the issues you've read about.

You should create your standards in the most user-friendly way possible. Probably the best examples are short, 10-slide presentations that show what everyone is supposed to do and not do. If you need to use a few more slides, it will still be okay. However, you should refrain from issuing a long document with lots of text. No one will read the document, and your proposed standards will not be adopted.

Getting to Know Your Features

If there is any weakness found in any site administrator, it's that they don't know what's possible. The fact that you're reading this book puts you ahead of the crowd and instantly raises your level of awareness to help administrate your website using as many best practices as possible. However, this book cannot go into the level of detail necessary to teach you everything that you need to do. It also can't teach you everything within the context of what's important to you and your organization.

In order to get to know what's possible and how DNN can best be leveraged to get the most value for your organization, you need to spend time getting to know the features. Try to have a local instance of the site to play with and then explore it.

Don't just look at the descriptions and labels and assume you know what's going to happen. Try something new. Check the checkbox and see what happens. Change that drop-down list you've always been wondering about and see what changes. Install the latest releases and take a look at the various new features. What do they do? Can you use them on your new site? Will any

of these things help you with the next phase of updates for your site? Can they reduce the amount of development necessary?

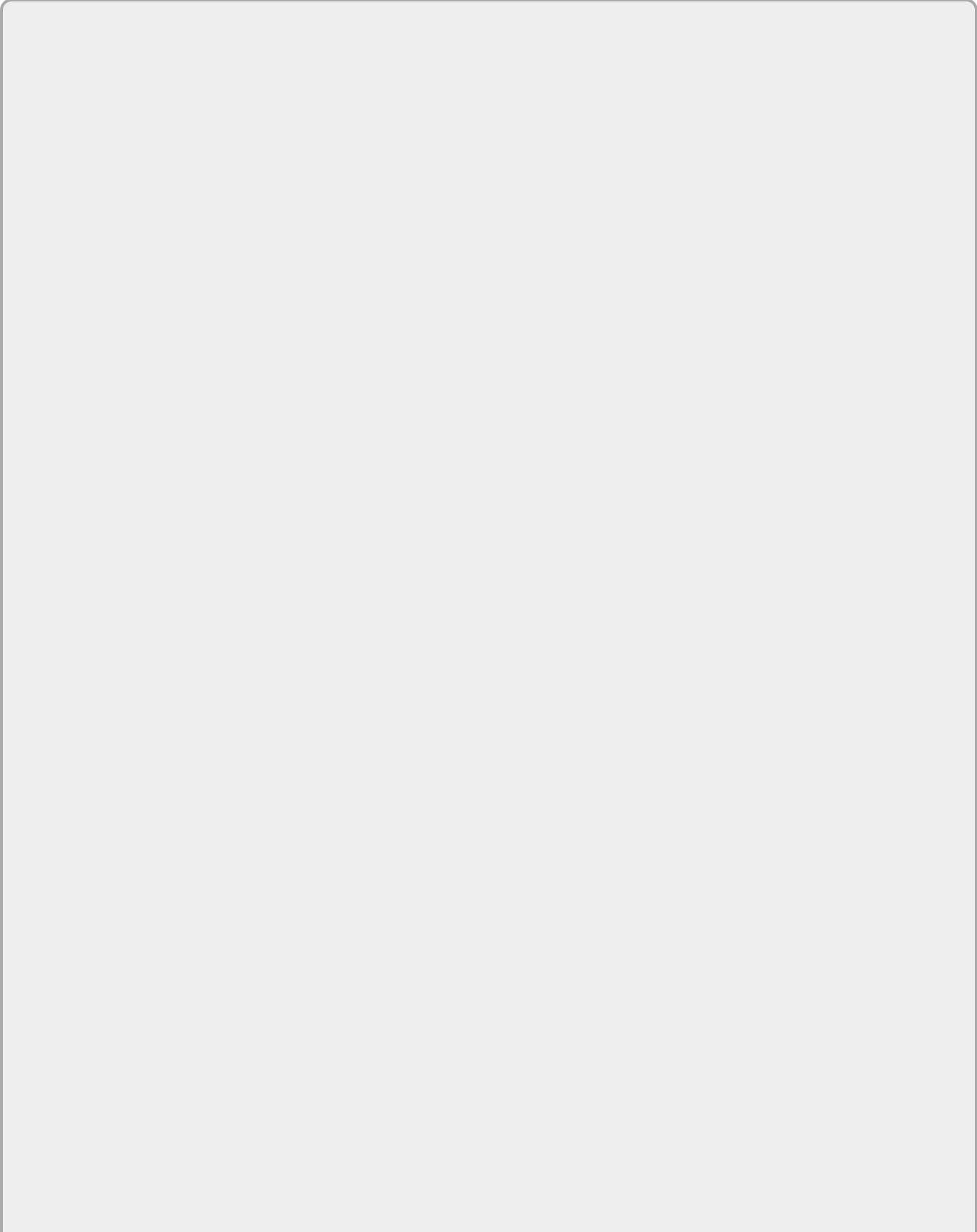
Don't be afraid...click away!

Summary

In this chapter, you learned what it means to be an administrator, and you received an overview of some of the most common features. In doing so, you've drilled down into DNN a bit more and highlighted key features that can reduce the time and headaches in managing your DNN site. We discussed numerous best practices and even went over some of the more common scenarios that you might run into. At this point of the book, you should feel more comfortable with the options at your disposal and how they will help you run the best DNN site possible.

Chapter 5

Host Administration



What You Will Learn In This Chapter

- Discovering the host's unique privileges
- Extending the DNN instance
- Changing sites and new site defaults
- Integrating with external services
- Entertaining development efforts

In [Chapter 4](#), “Site Administration,” you learned a great deal about administration of a site in your DNN instance. In this chapter, you're going to learn about administering the greater environment in which sites operate. The host user has the highest possible level of permissions in a DNN instance and can manage any individual site, set defaults for the creation of new sites, and manage additional configurations that support all sites. Most of these features are operational in nature, but some of them are oriented toward support of the developer community.

As the DNN host (or any user with superuser permissions), you need to have a working grasp of all the features and configuration options available to you. Throughout this chapter, you will learn about those options and features, gain insight into just how extensible and flexible DNN truly is, and learn to do a few things you didn't know how to do!

NOTE

DNN Corp. provides user and superuser manuals in the form of PDF files free from their website. An online version can be consulted at <http://help.dnnsoftware.com>. These manuals are extremely detailed and illustrate virtually every administrative page and function—content that we will not duplicate here. However, while these manuals detail what is available, they don't provide explanations as to why they are used, how they work, or what is wise or unwise to do. This is why you're reading this book, and the following pages focus primarily on those subjects.

Why Do You Need the Host?

This may seem a basic question, but its answer is an essential prerequisite for appreciating the significance of the information in the following pages.

One of the most powerful features of DNN is its ability to serve up multiple websites while existing as just a single application in IIS (Microsoft's Internet Information Services). When a traditional application serves only one website, the security model is pretty simple; one administrative user can do pretty much everything. But because a single instance of DNN can serve (or "host") multiple sites, it's organized around a security model that separates administration duties at the level of each site (or group of sites) and provides a higher level of security for actions that can impact other sites. Simply stated, site administrators cannot take actions that would impact other sites (such as recycling the IIS application domain) or otherwise change the runtime environment of the site they administer (for example, changing the application's `web.config` file) because that runtime environment is shared among all sites in the instance. Those privileges belong solely to the DNN host or the *superuser*.



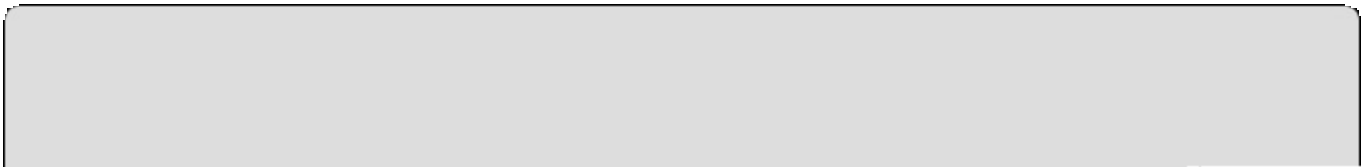
NOTE

From here on out, we'll simply refer to any user with superuser privileges as the "host." Although there are a few minor attributes unique to the designated host of a DNN instance, they don't impact operations that any superuser can perform.

What Is Host Administration?

You learned that a DNN site administrator is primarily concerned with the look, feel, functionality, and content of a site. A host is more concerned with the configuration, performance, monitoring, and support of the entire DNN instance.

A host logs into a DNN site just like any other user. But as you learned in [Chapter 4](#), users with different security roles can have slightly different views of a site, including its menus, Control Panel, and so on. This behavior applies to the host as well, revealing additional menus, menu options, and controls on various pages.



NOTE

Technically speaking, host privileges are not established by security roles as they are for administrators and other users. A host is actually a completely different kind of user recognized in code, although apart from access to some additional features they behave almost exactly the same. The `IsSuperUser` flag on the `Users` table denotes a host as opposed to an entry in the `UserRoles` table.

[Chapter 4](#) introduced the primary means by which administrative features are revealed, specifically the Control Panel and associated menus. There are a few additional places where host-level functionality is exposed, and we'll go through each of those so you know where to find all the features.

Host Attributes

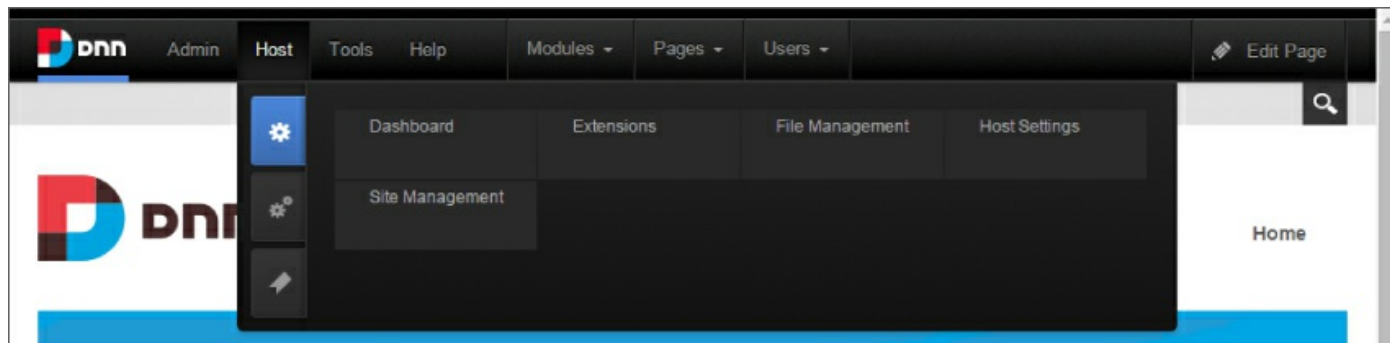
Before moving on, I want to clarify something. There is a difference between a host user and the designated “host” of a DNN instance. This chapter focuses on the host user (a superuser) and the things that user can do. A later section of this chapter also covers host details. These details don't have anything to do with a particular user; they have to do with designating one site in your instance as default and a few details such as the email address from which the host communications are sent.

NOTE

Host attributes are necessary to establish a distinction between attributes of your DNN installation and attributes of a particular superuser. These things are initially populated on install with the information collected to create the default superuser, but they are not associated.

Host Menu Pages

When the host logs in, the only immediately visible difference in the user interface is the Host menu option added to the Control Panel. See [Figure 5.1](#).



[Figure 5.1](#)

The Host menu has two sections, Common Settings and Advanced Settings. We'll go through each of the features in those menu sections in sufficient detail for you to utilize them. In later sections, you will revisit some of these features to consider more detailed use cases or to accomplish more specific objectives.

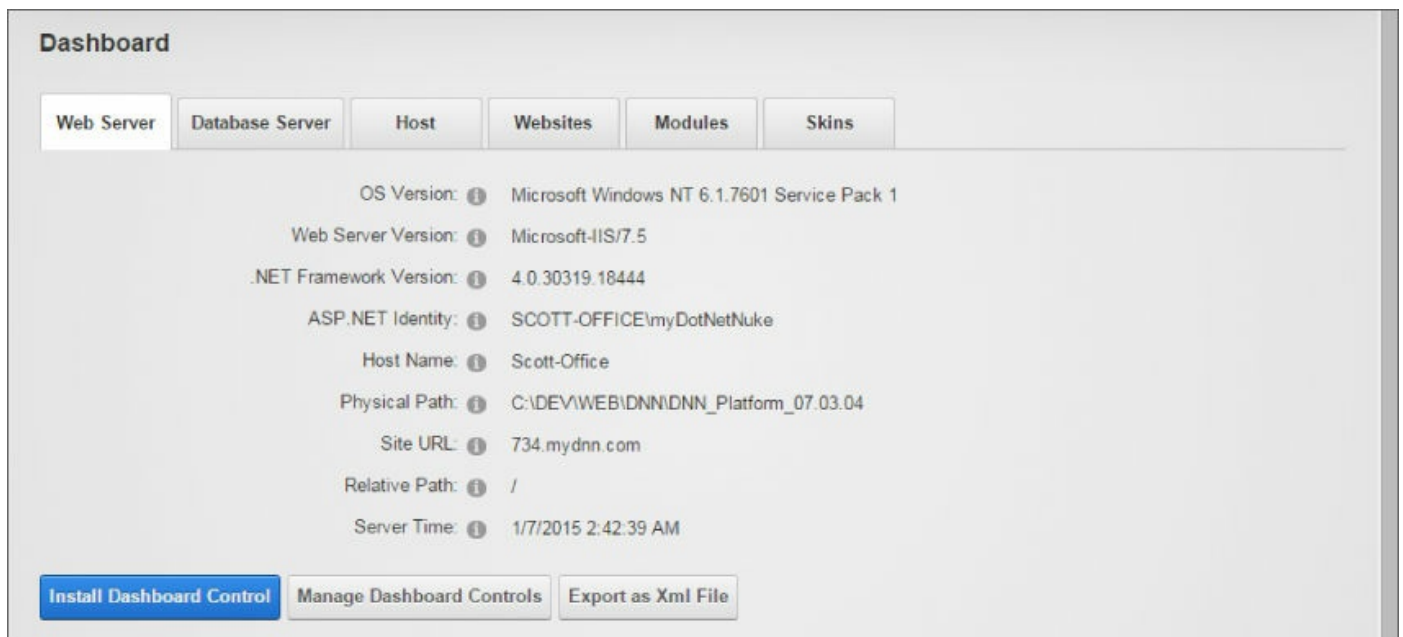


NOTE

Based on the previous tip, if you have direct access to the database, you can update the `IsSuperUser` flag for any user. This can be extremely handy during development if you forget the login credentials for a local install. Simply register a new user whose password you know and then flip the `IsSuperUser` flag in the database. Log in with that user's credentials, update the password of your forgotten superuser account, log in again with your restored credentials, and return the new user to normal by changing the `IsSuperUser` flag back to false.

Dashboard

The Dashboard page is exactly what it sounds like—an information center that provides a substantial amount of meaningful data “at a glance” about the DNN instance. It includes information gathered from the Web and database servers as well as specific DNN configuration information and listings of installed modules and skins. [Figure 5.2](#) demonstrates the default view of the Dashboard, opened to the Web Server tab on a test site, displaying detailed information about the configuration of that web application.



Dashboard

Web Server Database Server Host Websites Modules Skins

OS Version: Microsoft Windows NT 6.1.7601 Service Pack 1

Web Server Version: Microsoft-IIS/7.5

.NET Framework Version: 4.0.30319.18444

ASP.NET Identity: SCOTT-OFFICE\myDotNetNuke

Host Name: Scott-Office

Physical Path: C:\DEV\WEB\DNN\DNN_Platform_07.03.04

Site URL: 734.mydnn.com

Relative Path: /

Server Time: 1/7/2015 2:42:39 AM

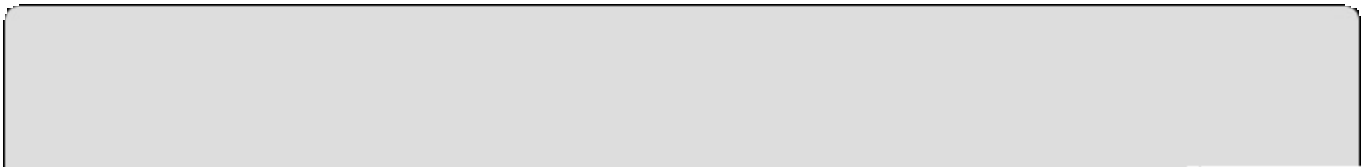
Install Dashboard Control Manage Dashboard Controls Export as Xml File

Figure 5.2

Dashboard information can also be exported directly into an XML file at the

push of a button as an aid for sharing information with technical support personnel. Exported files will be stored in the “host root” folder, which is a folder in your installation's directory structure called `<Root>/Portals/_default` (also accessible from the File Management page in the Host menu).

Like most aspects of DNN, the Dashboard was built with extensibility in mind, and it can be customized with the addition of more Dashboard controls. A new instance of the Dashboard can even have completely different controls. You can construct your own Dashboard control to convey the status of virtually anything, even querying outside systems if you want to consolidate your systems information in one easy place.



NOTE

Because Dashboard controls tend to be very personalized, there are not many examples available in the public domain. However, there is an open source project at <https://socialdashboard.codeplex.com> where author Bruce Chapman provides a coding example of a Dashboard control to display social activity in an installation. Bruce blogs about that example project here:

http://www.infinity.com.au/2012/09/21/New_DotNetNuke_Social_Dashboard

The Dashboard is designed to produce a static display of information, but that display may be a “snapshot” of a dynamic process. Consider, for example, an instance of the Dashboard with custom controls that are relevant to a business user or manager, informing her on status of multiple business processes like transactions in pending status, open/closed issues, stock levels, and so on. When there isn't a comprehensive reporting system or when users lack access or training on comprehensive systems, the Dashboard can provide a simple means of conveying essential status information to site users.

Extensions

The Extensions page is the entry point to a comprehensive system of acquiring, installing, managing, and even creating extensions of all kinds. For the sake of brevity here, this section focuses on understanding the Extensions page and identifying the extension types and functionality available to the host when managing them. But you'll come back to this issue later in this chapter with a working example.

Let's start by looking at the content of the tabs on the Extensions page.

Tab: Installed Extensions

[Figure 5.3](#) illustrates the layout of the Installed Extensions tab of the Extensions page. It contains a complete list of extensions of every type that are currently present in the instance. Because DNN is built with an extension model, its own features are also implemented as extensions. So even in a brand new instance, you'll notice that there are many extensions of many types already listed. Each extension type is displayed in a list that can be expanded or collapsed for increased readability.

Extensions

[Install Extension Wizard](#)
[Create New Extension](#)
[Create New Module](#)

[Installed Extensions](#)
[Available Extensions](#)
[Purchased Extensions](#)
[More Extensions](#)

[Expand All](#)

This application contains an Update Service which displays an icon when a new version of an Extension is available. The icon displayed will contain a visual indication if a currently installed Extension contains a potential security vulnerability. If a security vulnerability is identified, it is highly recommended that you upgrade to the newer version of the Extension. Clicking the icon will redirect you to a location where you will be able to acquire the Extension for immediate installation.

Modules

	Name	Description	Version	In Use	Upgrade?	
	AdvancedSettings		1.0.0	Yes		
	Authentication	Allows you to manage authentication settings	7.1.0	No		
	Digital Asset Management	DotNetNuke Corporation Digital Asset Management module	1.0.4	Yes		
	DnnChat	A Chat module for DotNetNuke using SignalR	1.0.0	No	01.01.00	
	Extensions	Allows a Super User to manage the various extensions such as Skins, Modules Language	7.1.0	Yes		
	Vendor	Allows you to add a new vendor, update an existing vendor, and delete a vendor.				
	ViewProfile		7.1.0	Yes		

Authentication Systems

Containers

Core Language Packs

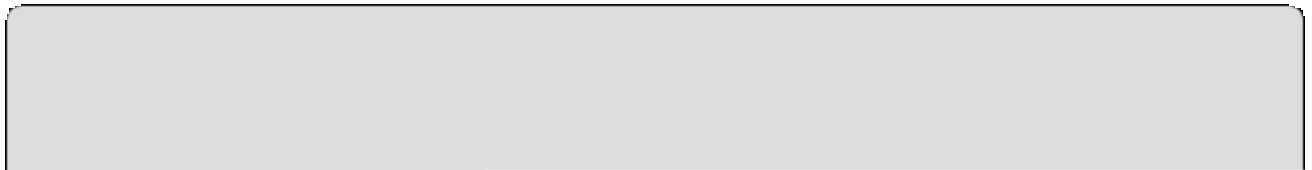
Figure 5.3

One helpful piece of information available on this page is the In Use report. If there is a Yes beneath the In Use column, clicking it will invoke a window that lets you select a site on your install to see what pages are using that module. Links to the pages are also provided. This is particularly helpful when you're considering module upgrades, replacement, or retirement

because it allows you to see the impact of the prospective changes.

DNN's extensibility model includes the following types:

- Modules are chunks of application functionality that can range in complexity from a simple image rotator to a complete Customer Relationship Management system. Modules are covered in more detail in [Chapter 6](#), “Modules.”
- Authentication systems provide the functionality that implements access security for a site. For example, permitting login by means of Facebook, Twitter, Active Directory, and so on. Authentication systems and security issues are explained in more detail in [Chapter 9](#), “Membership Security.”

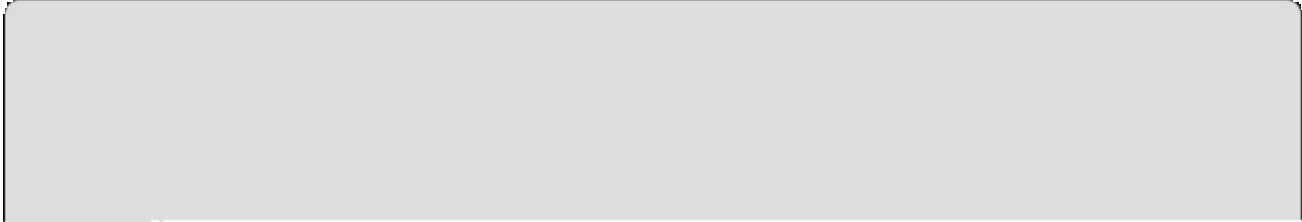


NOTE

Authentication systems are really just another type of “provider” and could have been listed in that section. The primary difference from other providers is the inclusion of UI components (for login and such). They have historically had their own section in the Extensions area for clarity.

- Core language packs contain all of the string resources to change the language of a base DNN install to another language. Note this does not affect user-developed content, just the language of the administrative interfaces and user-facing controls (the Login control).
- Extension language packs are very similar to core language packs, except they are specific to installed third-party modules. Consult [Chapter 10](#), “Localization,” for a comprehensive discussion of core and extension language packs.
- Dashboard controls, discussed in the previous section, are static information displays that can be plugged into instances of the Dashboard.
- JavaScript libraries are exactly what they sound like—third-party JavaScript libraries packaged for installation in DNN. Many JS libraries are available today for development, but rather than require developers to include them with each module, the site needs to support the JS library only once across the entire installation.
- Libraries refer to shared .NET assemblies. Like JavaScript libraries, they are shared across the entire installation and so are installed in such a way that they are not just associated directly with an individual extension that might leverage them.
- Providers include functional components that are implemented using Microsoft's Provider design pattern, which DNN utilizes in multiple areas. The areas currently supported by providers include: Caching, Client Capabilities, Cryptography, Data, Friendly URL, Folder, HTML Editor, Logging, Menu, Navigation, Permissions, Profile, Role, Scheduling, Search, and Sitemap. You'll learn about a few of these provider types in greater detail later in this chapter.
- Skins and containers are UI resources applied to sites and modules to

accomplish a specific look and feel or a “theme.” This subject is covered in detail in [Chapter 17](#), “Skinning.”



NOTE

Skins are installed with “scope.” A skin that is installed in the context of a site administrator is visible only to that site. The host can install skins that are available to all sites.

- Skin objects are .NET user controls that offer functionality that can be embedded in the skin of a site. This functionality is different than modules as it doesn't relate to content, but rather functionality delivered through the user interface. For example, breadcrumbs, search boxes, current date, and even rendering of dynamic menus are all accomplished using skin objects.
- Widgets are client-side “mini applications” written in JavaScript that can leverage installed JavaScript libraries to dynamically inject content into a skin or module UI.

NOTE

The widget framework was developed by Nik Kalyani, one of the original founders of DotNetNuke. His comprehensive four-part blog is still the ultimate resource for understanding and developing widgets. Read it here: <http://kalyani.com/2009/12/dotnetnuke-widgets-guide-part-1-of-4>.

Tab: Available Extensions

This tab is organized similarly to the Installed Extensions tab, with expandable/collapsible sections for resources that are available to install. “Available” has one of two meanings. Either a file is already “staged” on your instance and available for install or it is available on a trusted server and ready to copy to your instance and then install.



NOTE

*When DNN is initially installed, any resource file named *.zip and found in the <Root>/Install/<type> folders will automatically be included in the process. Files named *.resources will not be automatically included but are staged for future use.*

A “staged” resource is simply a file that exists in the proper location, specifically your <Root>/Install/<type> directory, with an extension of .resources or .zip. If you take a “deploy” action in any of the areas where it is available to you, files are deposited in these directories, which stages them for later installation at your convenience. The second meaning of “available” is that an extension has a trusted external host that is publishing the resource, which you can “deploy” and then install. Only core language packs (maintained by third parties on behalf of DNN Corp.) are distributed in this manner at the present time, which explains the availability of the Deploy action in this area (see [Figure 5.4](#)).

NOTE

In a local instance it would probably not be necessary to stage resource files (if you're running as a local admin). DNN's Install Extension wizard will permit selection of a resource file from any known disk location. However, in a production environment, it might be desirable to limit write permissions from the browser for non-user directories (that is, anything outside of <Root>\Portals). In this case, resource files must be placed in the staging area on the server either by direct server access or by using the Deploy method.

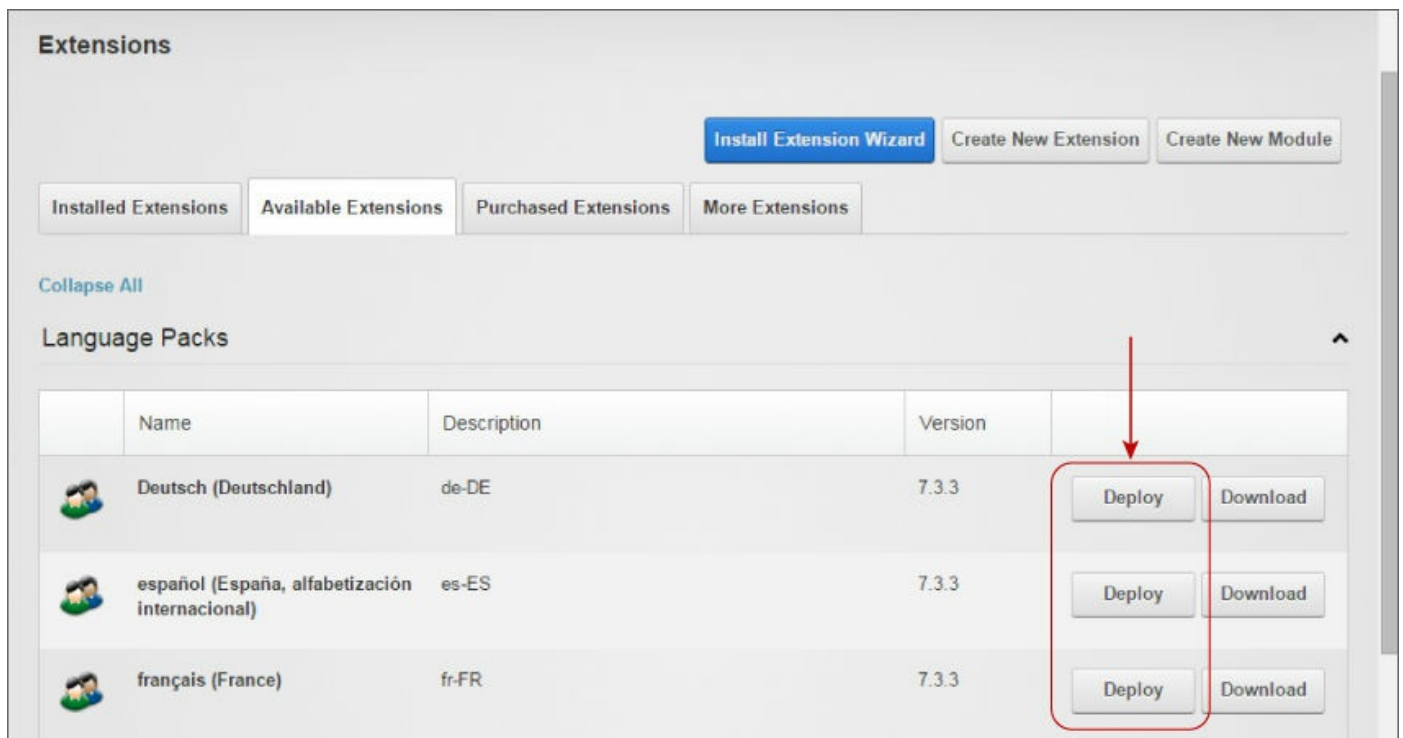


Figure 5.4

You can also retrieve staged files for local inspection using the Download button.

Tab: Purchased Extensions

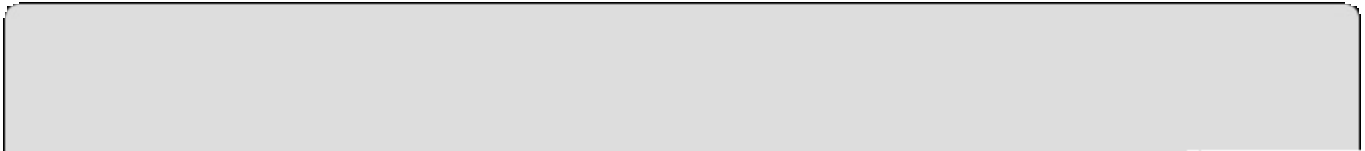
This area permits you to gain quick access to resources that you purchased in the DNN Store (<http://store.dnnsoftware.com>). This is covered in greater detail in [Chapter 22](#), “The DNN Store.”

Tab: More Extensions

As the name implies, features on this tab help you find more DNN extensions. This service is connected to both the DNN Store and the DNN Forge (the open source repository for community projects). This is by no means the only way to find DNN extensions, but many third-party vendors and open source developers take advantage of this service to promote their DNN wares.

Buttons: Install Extension Wizard

This button invokes the Install Extension wizard. It is not necessary for you to know what type of extension is in a resource file. DNN supports a unified manifest file in resource packaging, which permits the wizard to inspect the resource and install it appropriately. The manifest is covered in detail in [Chapter 18](#), “Packaging and Distribution.”



NOTE

*Staged resources (in the <Root>/Install/<type> folders) can be named either *.resources or *.zip, but the Installation wizard accepts only *.zip files.*

The wizard will progress through a number of dialogs during the installation process, including:

- **Upload New Extension Package:** Permits selection of the install file
- **Package Information:** Permits review of the extension definition as specified in the manifest, including type, vendor, version, and so on
- **Release Notes:** Permits review of any release notes specified by the vendor
- **Review/Accept License:** Displays the vendor-supplied license and an Accept checkbox, required to proceed
- **Package Installation Report:** Provides a text-based report of all the actions performed by the installer, ultimately indicating success or failure of the install

Buttons: Create New Extension and Create New Module

These buttons are used in support of development and are covered in detail in [Chapter 18](#).

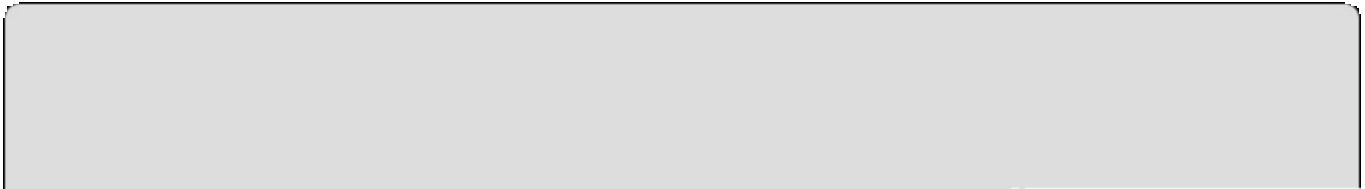
File Management

[Chapter 4](#) provided a comprehensive description of the File Management page and use of the Digital Asset Manager (DAM). The only difference between the administrator and host views of File Management is the root directory of the DAM. Where the administrator's "root" is associated with the site root directory (<Root>\Portals\<site dir>), the host's "root" is associated with the default site root directory (<Root>\Portals_default). This directory will contain user storage for superusers, site-creation templates, and some resources common to all sites. Two examples of this include host-installed skins and containers (visible to all sites) and, in a default instance, a folder called <Root>\Portals_default\Smileys and utilized by the

RadEditorProvider.

Host Settings

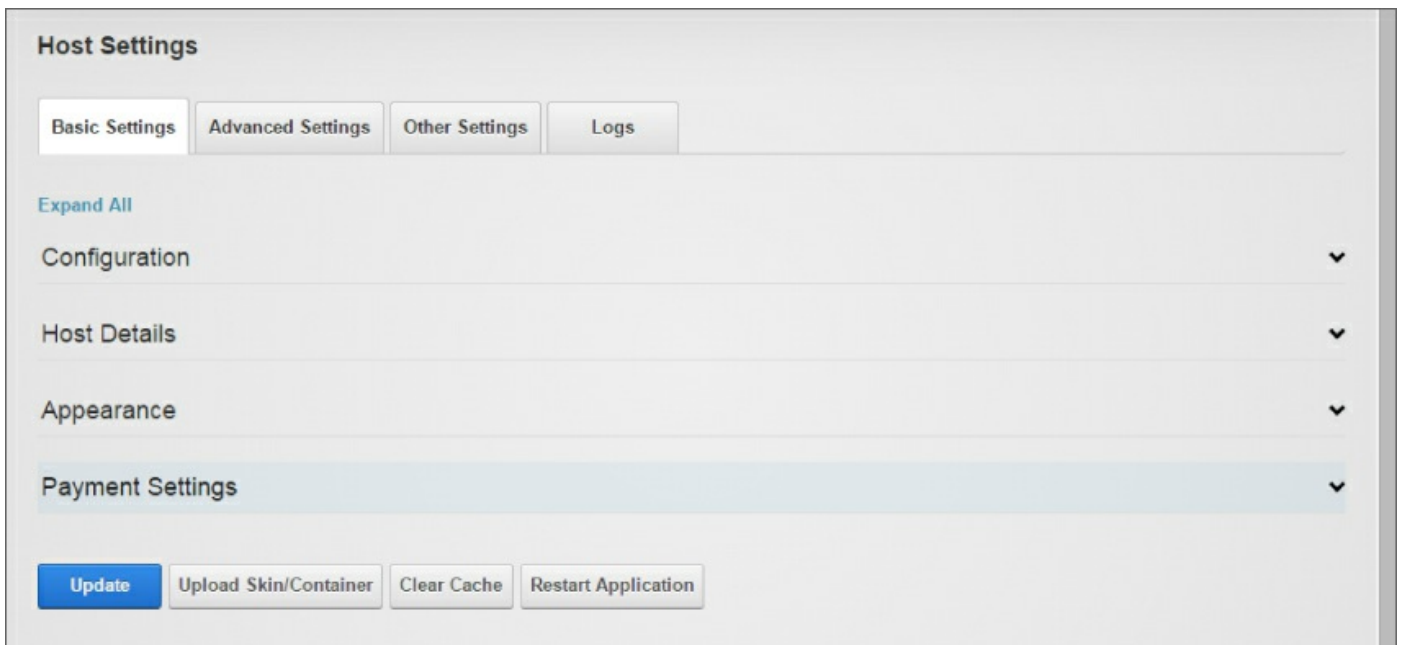
This is the primary menu from which the host will perform configuration for the DNN instance. The settings represented here are not configured on a per-site basis (except where noted). They are configured for the instance overall, affecting all the sites they contain.



NOTE

We're not going to cover every setting. Many DNN settings are self-explanatory and/or the associated help for them is comprehensive. We do, however, provide explanations for how some things work, non-obvious impacts of setting choices, and use cases where they may not be obvious.

In [Figure 5.5](#) you'll see that the Host Settings page has four tabs and four buttons. Each tab is further organized into a series of expandable and collapsible sections containing relevant options and settings.



[Figure 5.5](#)

The Buttons

The page's default button is Update, which performs as expected, saving the changes you have made to any settings. If you don't click Update, your changes will not be saved. And if you do click Update, depending on the changes you've made, you might force an application restart. If you're in a production environment, that's not something you'll want to do without a little care, planning, and notification to users.

The Upload Skin/Container button invokes the Install Extension wizard covered in the previous section. It's provided mostly as a convenience, since

clearly it doesn't restrict your uploading to just skins and containers. Be sure, however, to recall that skins and containers uploaded by the host are visible to every site. So if your intent is to upload only for a specific site, you'll want to do that from the Site Settings, not from the Host Settings.

The Restart Application button causes DNN to “touch” the `web.config` file (no changes are made, but the file modification date is updated). IIS responds to this by restarting the application domain of the site in question. The effects of an app domain restart include reloading and “re-JITing” all of the assemblies that make up the application as well as loss of any in-process variables (session, application, and cache). Since DNN doesn't use session state, that's OK, but the application and cache values must be rebuilt on the restart. Some of this impact could be partially mitigated by removing state information to an out-of-process option (such as `StateServer`) and alternative cache mechanisms (such as `AppFabric`).

The Restart Application button can be particularly helpful when a site is behaving strangely and an errant application is suspected, especially when you are in development. However, in a production scenario, using this button is staunchly frowned upon, as an App Restart is highly disruptive to users. Not only will all users be disconnected from every site in the instance, but any in-process context information (the default configuration in DNN) will also be lost.



NOTE

Just to be clear, Restart Application is like the nuclear defense option of production maintenance. In a large website with many users and assemblies, a full restart and repopulation of cache could take several minutes...a lifetime in web operations terms.

The Clear Cache button instructs DNN to unload all “non-essential” data currently in the cache. Non-essential is relative to the application, referring specifically to cached items that are not part of the configuration. So in general, performing a Clear Cache operation should not adversely affect the operation of the site, although it can adversely affect some users who might be in the middle of multi-step processes (where cache is essentially holding their current “state”) and all users in terms of site delay where previously cached objects need to be retrieved from the database again.

NOTE

Consider that some things that are harmless in a development environment can be quite disruptive in production. For example, the impact of clearing the cache on a developer machine is almost nil, as IIS is processing only one user's requests. However, in a production instance there might be thousands of requests in the queue, which would cause the application to try to rebuild virtually all of its cached items at once. This could result in resource bottlenecks, CPU spiking, and poor responsiveness for users while the site tries to go from “0 to 60” in an instant.


Although the number of items unaffected by Clear Cache can vary slightly, it is typically less than 100. Given a few seconds, that number will climb to about 300 as various helpful items are loaded into cache, such as site navigation structure and menus, CSS and JS files, various additional host, site, page, and module settings, and so on. But on a large and/or busy site, cache can easily grow to a few thousand items or more. Multiply this for each web head if a web farm is involved.

NOTE

The professional edition of DNN, called Evoq, has an additional host feature that reveals current memory usage and permits inspection and selective expiry of cache items.

Basic: Configuration

Like the Dashboard, Configuration reveals some interesting information about the current configuration, including DNN version, physical path on the file system, identification, permissions of the ASP.NET identity for the site, and so on. It has only one setting—Check for Upgrades. What exactly does this button do?

DNN releases new versions of its software fairly regularly with various fixes and minor enhancements, and it can be very helpful to know when new versions are available. This setting affects not only the DNN Platform notification on this page but also the new version notification for entries on the Host  Extensions page. It does this by pinging back to DNN the version that you are running and the extension versions that are installed. This pingback does not contain any personally identifying information, configuration, or content; it is purely for comparative purposes (much like other applications that gather usage data to aid in improving the app experience).

In a managed environment, such as a production installation, this feature is not necessary and should be turned off; updates are planned, scheduled, and practiced. However, during development the usefulness of new version notifications far outweigh any perceived cost (the occasional pingback and response).

Basic: Host Details

As discussed earlier, host “details” are not associated with a user. The attributes (Title, URL, and Email) are most commonly surfaced in an email sent by the site for things like notification of new superuser accounts, sending of host information (such as logs), or through various skin objects.

NOTE

Host details are infrequently used but can be helpful. Consider the case of a small business that provides hosting for customer sites through one or more DNN installs. These details can easily be referenced in skin objects to reveal contact information for a “Powered By” type link.


The checkbox for Enable Remember Me does as you'd expect, enabling this option on the login window. It is a host setting because security and authentication configuration is implemented across an entire instance rather than on a site-by-site basis.

Basic: Appearance

The Show Copyright Credits option is a legacy setting that impacts only certain skin objects and injection of a comment in the HTML source of your rendered pages. You should uncheck this option to make it less obvious to spiders or others what application your web server is running.

Unchecking Use Custom Error Messages is advisable only for development environments, although even in these conditions you might choose to leave the option enabled. It simply suppresses raw ASP.NET error handling in favor of more gracious error handling to the user. Admin and host users will still see the inner stack trace message. If you disable custom error messages, DNN will not catch the exception, and it will bubble up to the application level, at which point it will respect the `customErrors` settings in `web.config`.

NOTE

Don't expect this setting to have any impact on capturing 404 Page Not Found errors. DNN includes a special 404 Page not Found page, which you can update in Admin  Page Management.

Basic: Payment Settings

This is one area of legacy functionality that we discourage you from using. The DNN Platform is simply not an ecommerce system; however, there are many available extensions to provide ecommerce capability. The few settings available specifying hosting space and page and user quotas can be somewhat helpful, as they are enforced on newly created sites. However, we're more fans of managing these items relationally with clients rather than through hard limits, which can impact site experience.

Advanced: Friendly URL Settings

It's important to know that most of the rules provided in the default instance have historical value relative to upgrades. They help redirect links that worked in older versions of DNN to their newer DNN equivalents. For example, the rule `[^?]*\/TabId\/(\d+)(.*)` specifically maps an old URL “machine-friendly” format back to an internal format that any new friendly URL provider can utilize.

If you're not planning to install any enhanced URL management providers, you can do some basic URL crafting here, although an understanding of Regex (regular expressions) syntax is required. However, it should be noted that friendly URLs are not a substitute for advanced URL management; they have no support for situational routing, nor any concept of URL indexing, and so on.

The underpinnings for advanced URL management were added to DNN in version 7.1 and are exposed in the Evoq professional offerings. However, new third-party extensions leveraging that built-in capability should be forthcoming soon. One example of this is a community-supported open source project called DnnUrlManagement (<https://dnnurlmanagement.codeplex.com>).

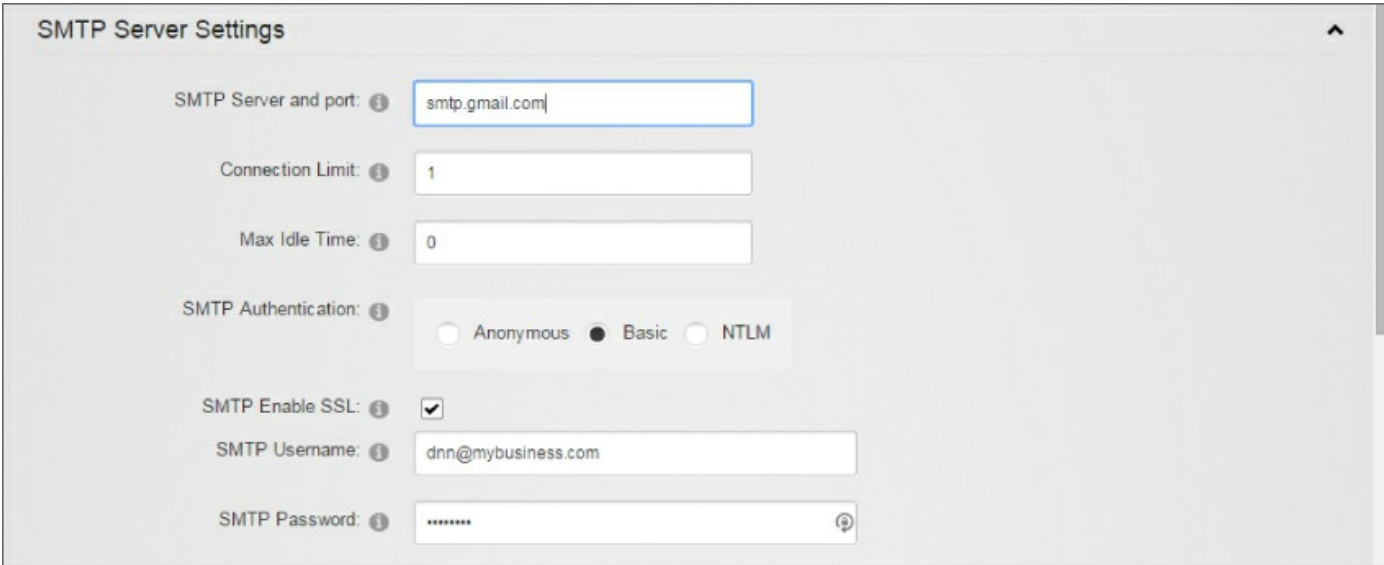
Advanced: Proxy Settings

Depending upon your configuration and usage of third-party modules, the DNN instance may be reliant upon some server to server interconnectivity. Examples of features that might display this reliance include the critical updates notification in Host Settings ⇅ Basic Settings ⇅ Configuration or when referencing RSS feeds, FTP, NNTP, or web services. If your DNN instance is separated from a direct connection to the Internet by a proxy server, you should consult with your network administrator about the appropriate settings.

Advanced: SMTP Server Settings

The DNN instance sends outbound email for a variety of reasons, from the host or site admin or in the form of password reminders, event notifications, newsletters, or other business-specific use cases. A valid SMTP server is required for this to work; you should consult your network administrator for the appropriate credentials to use in a production configuration.

Most non-Exchange Server SMTP hosts will require “Basic” authentication and a valid username/password combination. Some will also require that SSL be enabled. For example, [Figure 5.6](#) illustrates a proper configuration for a small business using Gmail as part of Google Apps for Work. If you are in a development configuration and your ISP does not permit access to port 25 (the default for SMTP), an alternative port can be specified (such as `smtp-mail.outlook.com:587`).

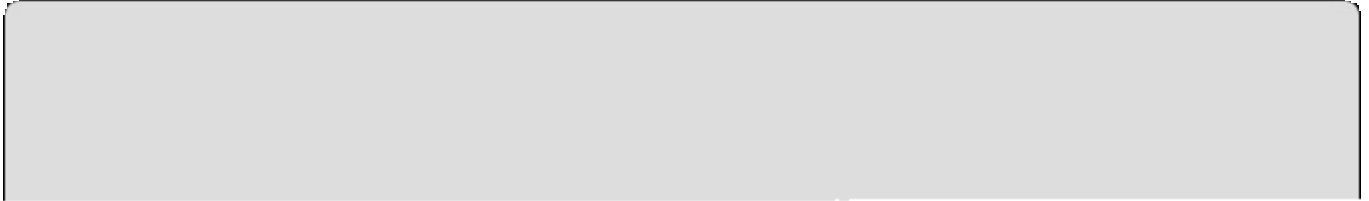


The screenshot displays the "SMTP Server Settings" configuration interface. It includes the following fields and options:

- SMTP Server and port:**
- Connection Limit:**
- Max Idle Time:**
- SMTP Authentication:** Radio buttons for Anonymous, Basic (selected), and NTLM.
- SMTP Enable SSL:**
- SMTP Username:**
- SMTP Password:**

Figure 5.6

This setting is for the host SMTP server and the default for new sites. You should note that the host can also set up SMTP server settings for each site, if preferred. This setting is located in the Admin [↗](#) Site Settings page but is visible only to a superuser. It can also be reached from the Host [↗](#) Site Management feature.



NOTE

Although it's beyond the scope of this book, it is essential that email coming from your site be properly identified by recipients to avoid the dreaded SPAM filter. Your site DNS should contain properly formatted PTR, SPF, and DKIM records for your domain.

Advanced: Performance Settings

The Memory option for Page State Persistence is unreliable in combination with Ajax; it's even noted in the user interface. This might seem unfortunate because it means that your browser payload is going to include a block for `_VIEWSTATE`, but there is good news! First, the use of view state is 100% reliable. Second, and more importantly, recent versions of DNN have undergone massive efforts to reduce the size of view state, resulting in a payload of about 160 bytes (yes bytes, not kilobytes) for an anonymous user and generally less than 1KB for a user who is logged in. This is nominal payload impact, even at its worst. The only reason this option originally existed was to contend with the size of view state; expect it to be deprecated in some future version.

The out-of-the-box choices for Module Cache Provider are Memory (the default) and File. Memory utilizes the standard ASP.NET in-process cache object. The File-based caching provider utilizes a custom approach, relying on an ASP.NET File System watcher to trigger updates during runtime. Traditional wisdom says that Memory is the better option unless you are resource-constrained.

The Cache setting affects how long objects remain in cache when unused (the longer they are in cache, the less likely they are to be reloaded). Again, unless you are resource-constrained there is no reason not to rely on “heavy” caching. Some modules also override these settings, as caching opportunities are limited for dynamic content.

NOTE

The settings for module caching can actually be overridden at the module level. So, if it makes sense to keep some large static text content in File-based cache rather than memory, you can simply override this on the Module settings. An interesting side effect of File-based caching is that, unlike Memory, it can persist between restarts (depending on the expiry settings).

Authenticated Cacheability is an ASP.NET feature that refers to whether a page can be cached on a downstream device during the page response lifecycle. Devices that can cache pages include the browser, web server, and any other cache-capable devices, such as proxy servers, and so on, between the web server and the client. The default setting, `ServerAndNoCache`, simply enforces that only the web server hosting the DNN instance is permitted to maintain the cache.

Advanced: jQuery Settings

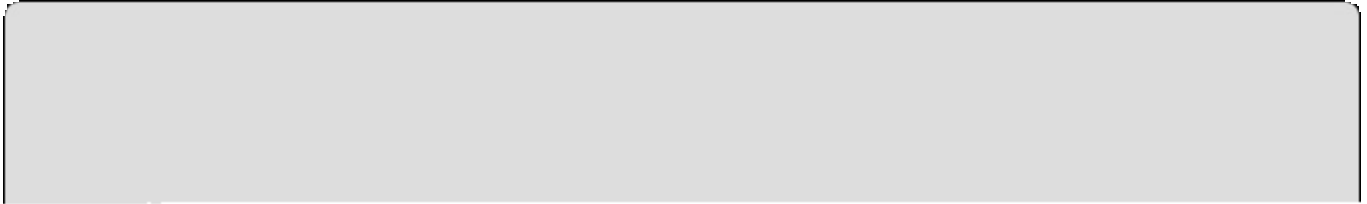
jQuery libraries are included with DNN for reference by extensions, whether in production use or in development. This information panel identifies the jQuery version and jQuery UI version that are currently active in the instance. Libraries periodically change. They are sometimes updated, expanded, or patched at unannounced intervals. DNN provides a means by which the host can always point to a “current” jQuery version rather than the installed one (if so desired). We'll cover that in the following section on CDN Settings.

Advanced: CDN Settings

CDN stands for Content Delivery Network. The CDN settings determine how your site will incorporate certain content that supports CDN options. By default, a new DNN instance will include its own local copies of specific versions of the MS Ajax, Telerik, and JavaScript libraries.

Enabling the MS Ajax CDN will instruct DNN to refer to those hosted at ajax.aspnetcdn.com. For more information on the MS Ajax CDN, refer to <http://www.asp.net/ajax/cdn>. The JavaScript Libraries CDN option behaves similarly, retrieving libraries from CDNs hosted by googleapis.com and jquery.code.com, with fallback to the local versions.

Telerik also maintains a CDN for its resources (JavaScript, CSS, and images), which is the DNN default. However, if desired, DNN also supports the ability to override that CDN and point to a custom URL. This might be desired if, for example, you were supporting a development team utilizing a pre-release version of Telerik UI for a future deployment.



NOTE

There are distinct advantages to referencing reliable CDNs in a production environment. CDN contents are cached on servers around the world, which means they may be able to reach a user's browser much faster than the copies on your web server. In addition, a CDN enables browsers to reuse cached third-party JavaScript files that may have already been loaded while browsing other websites. On the other hand, no CDN can match the delivery performance of a local intranet, and any CDN reference means that a live Internet connection is always necessary. Consider your relevant trade-offs.

Developers should be aware of which libraries they are building against and which deployment targets they are building for.

Advanced: Client Resource Management

Client resources (stylesheets, images, scripts, and so on) are one of the primary sources of poor client-side performance, so anything you can do to improve that is a good thing. Because DNN is so modularly constructed, many resources are provided in “chunks” relative to the modules in use. Without optimization, this results in multiple resources referenced on the page and multiple HTTP requests for those resources. Client Resource Management options provide mitigation of this issue through the use of three methods—versioning, compositing, and minification.

The Enable Composite Files option combines the contents of disparate files of the same type into a single file (or in some cases, just fewer files), which reduces the number of HTTP resource requests made by the page. By way of comparison, loading the home page of a default instance while logged in as host generates references to 13 CSS files, reduced to one when compositing is enabled. Note that authenticated user page requests generate more resource references than unauthenticated due simply to the additional page controls.

Script and stylesheet files often contain unnecessary whitespace and/or comments. Although they might be helpful to the developer, they have absolutely no value to the browser and can be safely “removed.” The Minify CSS and Minify JS options perform this function for their associated file types. [Figure 5.7](#) illustrates the CRM section with all of the options in effect.

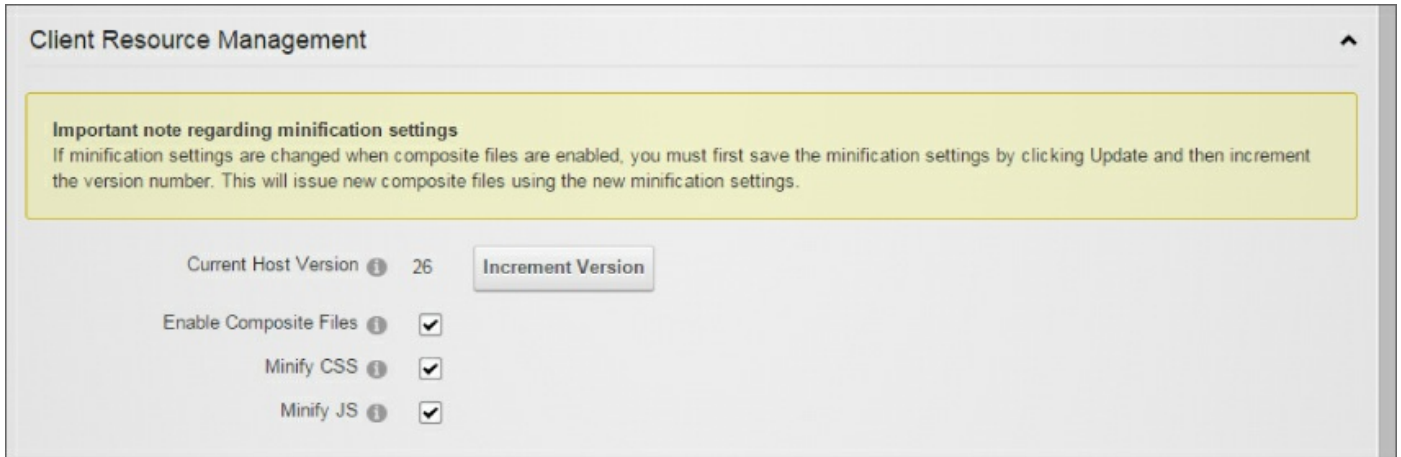
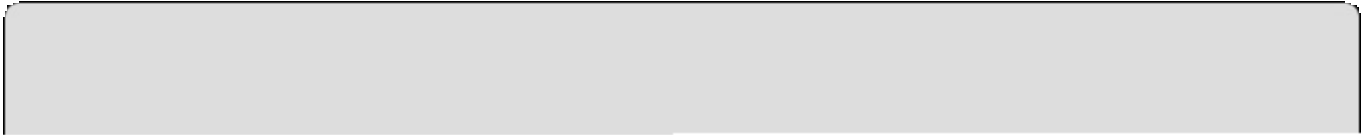


Figure 5.7

Finally, DNN also supports versioning of composited/minified files. This is necessary for forcing downstream devices (including the browser) to reload resources that it might otherwise not recognize as having changed. Some operations cause the CRM version to increment automatically (such as updating the `portal.css`), but it can also be done manually as necessary.



NOTE

CRM is awesome but not foolproof. Compositing and minification can have adverse effects on some modules; JavaScript in particular can be susceptible to errors from compression. A host should verify in a staging environment that the positive impacts of these features are not outweighed by errant behavior anywhere in the site. Similarly, developers should test their stylesheet and JavaScript code for CRM safety.

Advanced: Membership Management

In this area, the host can set parameters governing authentication. These settings apply to all sites, as changing the security parameters for one site would effectively change the security status of the entire DNN instance.

There are two settings for timeout values related to password-reset links. The Reset Link Timeout setting refers to user-initiated password resets and is typically a lower value, as it assumes that the users will be anxiously awaiting the reset they've just requested. The Administer Reset Link Timeout setting refers to password changes initiated by the host or administrator. These values are typically longer, as they might sit in a user's inbox a while before they are received.

NOTE

All of the password-management features (reset, expiry, history, ban, and strength check) are applicable only to DNN's standard authentication. If you're using a different authentication provider such as Facebook, these attributes are effectively outsourced to that provider.

The checkbox for Enable IP Address Checking must be enabled for Login IP Filters to be in effect.

Advanced: Login IP Filters

Login IP Filters enable creation of rules to permit or deny host/admin access by IP or IP range. Users attempting to log in will be greeted by a system message indicating that login is not permitted from their IP address.

IP filters can be used to enforce simple physical security rules that might, for example, limit administrative access to a particular subnet within an intranet. The checkbox for Enable IP Address Checking in Membership Management must be enabled for Login IP Filters to be in effect.

Though it might be tempting to utilize this feature to limit access during an upgrade process, this configuration is untested and not advised.



NOTE

The IP Filters feature has been tagged for deprecation in a future release, so it should be avoided. The Request Filters feature is capable of handling this case by examining the `REMOTE_ADDR` server variable. Similarly, IIS request filters can be coded directly into the `web.config` file via Host Configuration Manager.

Advanced: Search Settings

Lucene-based search capability was added to DNN in version 7.1. Administrators can now manage a great deal of search behavior for their particular site; see [Chapter 4](#) for information. There are, however, still a couple of search-related details that are managed at the host level.

An in-depth discussion of Lucene is beyond the scope of this book, but it is important to understand that it utilizes “analyzers” to parse content for indexing. DNN contains a variety of analyzers, including a number of foreign language parsers, defaulting to the general-purpose “Standard Analyzer.” Two additional interesting general-purpose analyzers are also included—the Whitespace Analyzer, which simply separates content tokens by whitespace, and the Stop Analyzer, which removes common English words typically not useful for indexing (although this can also be somewhat accomplished by an administrator utilizing the Ignore Words option).


Since search data is logically segmented by site, the Re-Index Host Content button literally pertains just to content that is found under the Host menu. However, the Compact Index option is available only to the host, as it affects the physical data store containing all of the site indexes. If you have a large site (or many sites), this is an operation that you would be wise not to perform during peak hours. It is resource intensive and can be invasive to site users.

Other: Request Filters

Request filters provide a mechanism for intercepting and redirecting page requests based on any criteria that can be found in the request header. This can be powerful and helpful functionality but should be used sparingly. Recall that there are other processes that also act on page requests that might be

affected by your rules (such as friendly URLs, advanced URL management, device detection, and so on).

One common use of Request filters is temporary blocking of malicious users by IP address. While this is not a permanent solution, a quick Request filter can thwart a bot, thereby giving you time to work with network support on a more appropriate solution. This is accomplished by inspecting the `REMOTE_ADDR` server variable for the IP in question (or using Regex to discern a range of IPs).

Be advised that this kind of functionality is provided in great depth by IIS via the `requestFiltering` node in `web.config`, which can be directly edited from the Host  Configuration Management page.

Other: Site Log

The Site Log and associated settings should be left at their default values, which essentially turn it off. This legacy functionality will be deprecated soon.

Other: Auto-Unlock Accounts After

This feature impacts user authentication but is not part of the authentication provider interface, hence its location in the Other Settings area.

Secure authentication contends with brute force attacks (password cracking) by disabling user accounts with a certain number of failed login attempts in a given time period. Note these settings can be changed only in `web.config`, as illustrated in [Figure 5.8](#).

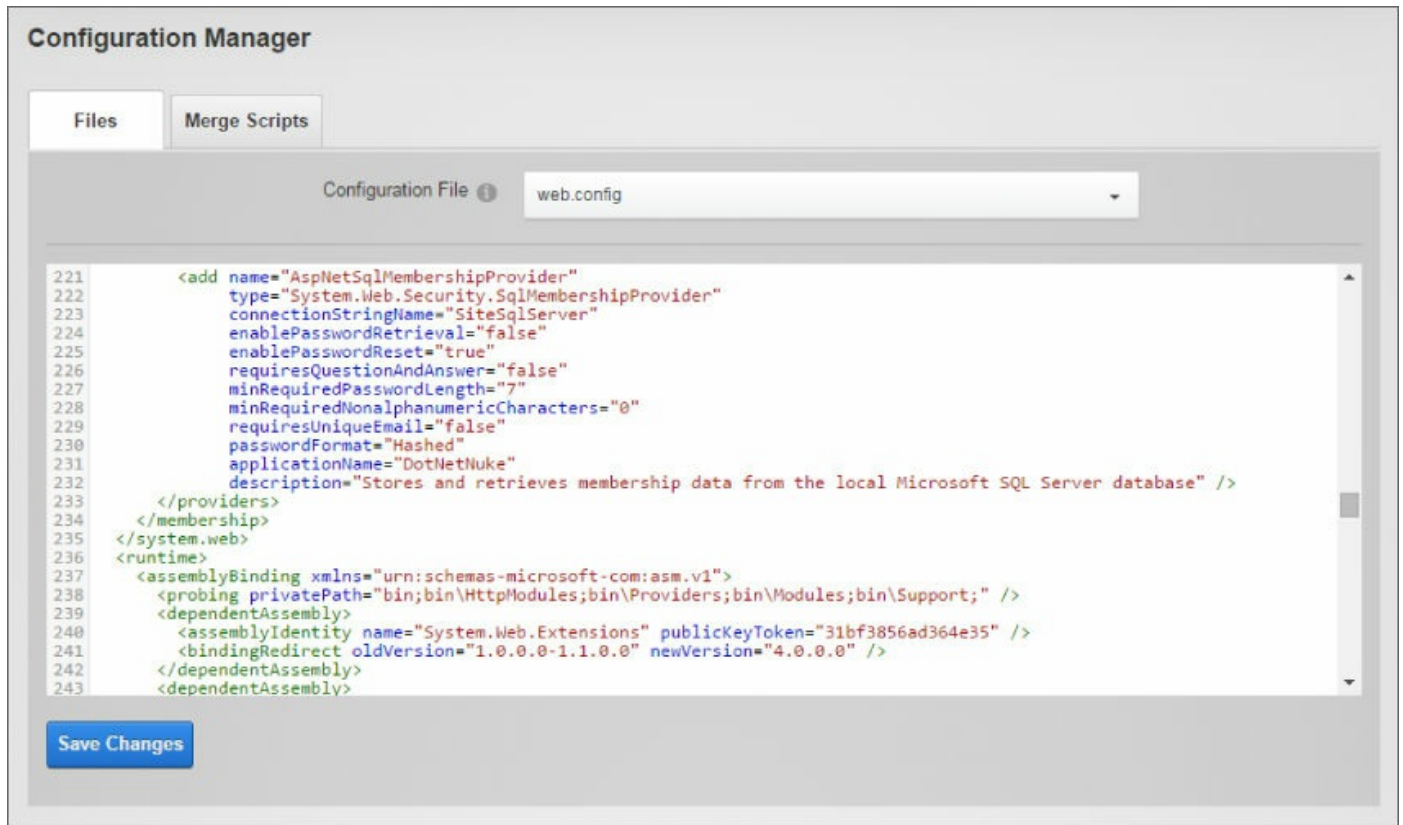


Figure 5.8

This feature provides a means of recovery for users from this “locked” state, enabling them to attempt to log in again after a certain amount of time has elapsed since their last failed login attempt.

Other: Allowable File Extensions

As mentioned in [Chapter 4](#), the Allowable File Extensions option functions as a white list for file handling in DNN. Files cannot be uploaded to DNN unless their extension type is included in this list. Similarly, if files are placed manually on the file system, they will be ignored by the Digital Asset Manager and Auto-Sync unless their extensions are present on this list.

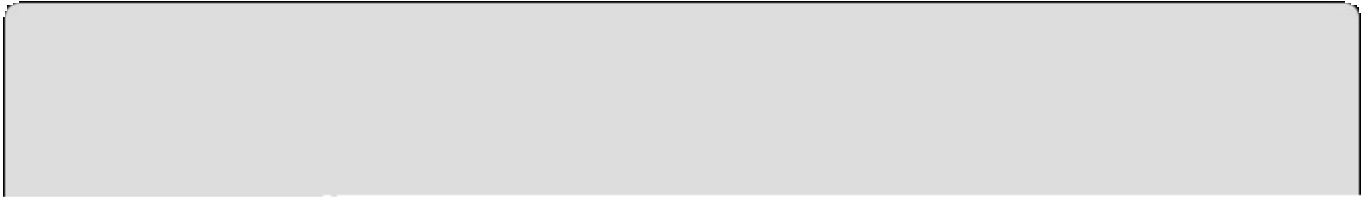
Other: Auto-Sync File System

If files are added to the file system using DNN features (such as uploading from the DAM, adding a profile picture, and so on), they are handled at that point in time by the application and properly recognized. However, files added by other means (such as FTP, direct server access, and so on) are not immediately recognized. The process of resolving this discrepancy is called “file synchronization” and can be performed manually from the DAM or

automatically by DNN on a periodic basis.

Other: Allow Content Localization

[Chapter 10](#) explains the concepts behind this seemingly innocent button in depth, but this section gives you enough information to convince you of the potential impact of this setting.



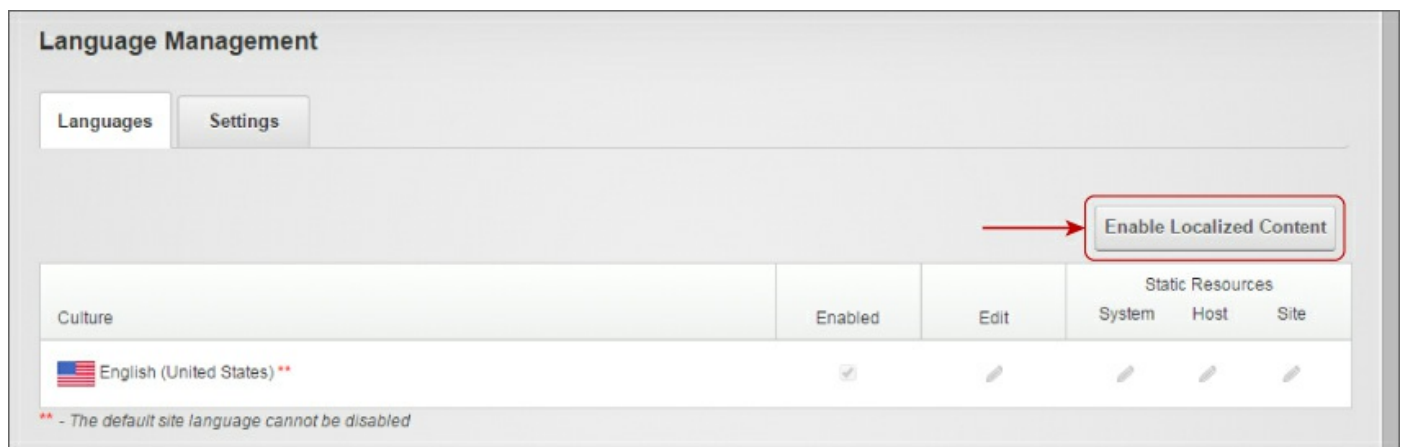
NOTE

This is one little checkbox with one very large impact. Before changing this setting, be 100% confident that you know what you are doing because it enables site administrators to perform actions that cannot be undone!

There is a difference between core localization and content localization. Core localization provides for multiple languages in the host, admin, and public-facing controls of the DNN application and is basically accomplished by including additional resource files and switching between them.

Content localization refers to the translation of user-supplied content in the site, for example, the verbiage on the typical About Us page. Multiple languages are supported by creating new versions of every page for each additional language, beneath a language “branch” (such as `/en-US/AboutUs`). Once the decision to change this site structure is made, it cannot easily be rolled back.

[Figure 5.9](#) illustrates the additional functionality that is exposed to the site administrator by enabling this feature.




[Figure 5.9](#)

The Enable Localized Content button pictured in [Figure 5.9](#) converts the site to use the localized branch structure.

Site Management

The Site Management page provides an inventory of all the sites in a DNN

instance and quick access to the Site Settings functionality for each. In the default install, the site settings accessed this way appear in a modal dialog as opposed to the normal page rendering when accessed through the Admin  Site Settings menu.

Additional functionality on this page includes creation and deletion of sites as well as the ability to export a particular site as a template. The Export Template feature is useful for creating your own default site template, with options to include or ignore various resources.

The host's view of the Site Settings menu is different than the administrator's view, with many more site-specific features. We'll go over these features in detail in another section later in this chapter.

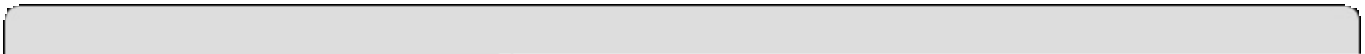
Configuration Manger

This feature puts every DNN configuration file at your fingertips in a capable editor with syntax highlighting. It also includes a utility for merging XML scripts, which is not only a handy maintenance tool but is also an integral part of DNN extension packaging, which you'll learn about in [Chapter 18](#).

An example of merge scripts can be found in the `<Root>\Install\Config\` directory. The `Net40.config` file was executed by DNN at the time of setup to adjust default environment settings for use with ASP.NET 4.0 (if required).

Device Detection Management

Device detection was added to DNN in version 6.1 to facilitate adaptive design and provide increased support for mobile devices. DNN Corp. partnered with 51Degrees (<http://51degrees.com>) to embed a Lite (free) version of their managed device library and a simple version of their detection API for use in DNN. Device detection must be enabled by the host to function in any site in the instance.



NOTE

51Degrees offers premium and enterprise editions of this library, featuring over 100 properties for over 20,000 devices and daily updates of the device database.

Only the Width, Height, and User Agent properties are exposed in the free version. This information is sufficient to establish a reliable mobile solution, but for finer grain adaptive design, the device properties available in the premium version are highly desirable. DNN's partnership with 51Degrees adds significant capability to the platform and is a great option for easily improving on it, if necessary, for your specific purposes.

HTML Editor Manager

The HTML editor, like most things in DNN, is implemented using the provider pattern. So the functionality exposed on this page will be subject to the capabilities of the underlying HTML editor and the features exposed through the specific provider. By default DNN will be configured with the RadEditorProvider, which implements the Telerik Rad Editor.

[Figures 5.10](#) and [5.11](#) illustrate some of the differences between settings available for the Telerik Rad Editor and the CKEditor, respectively.

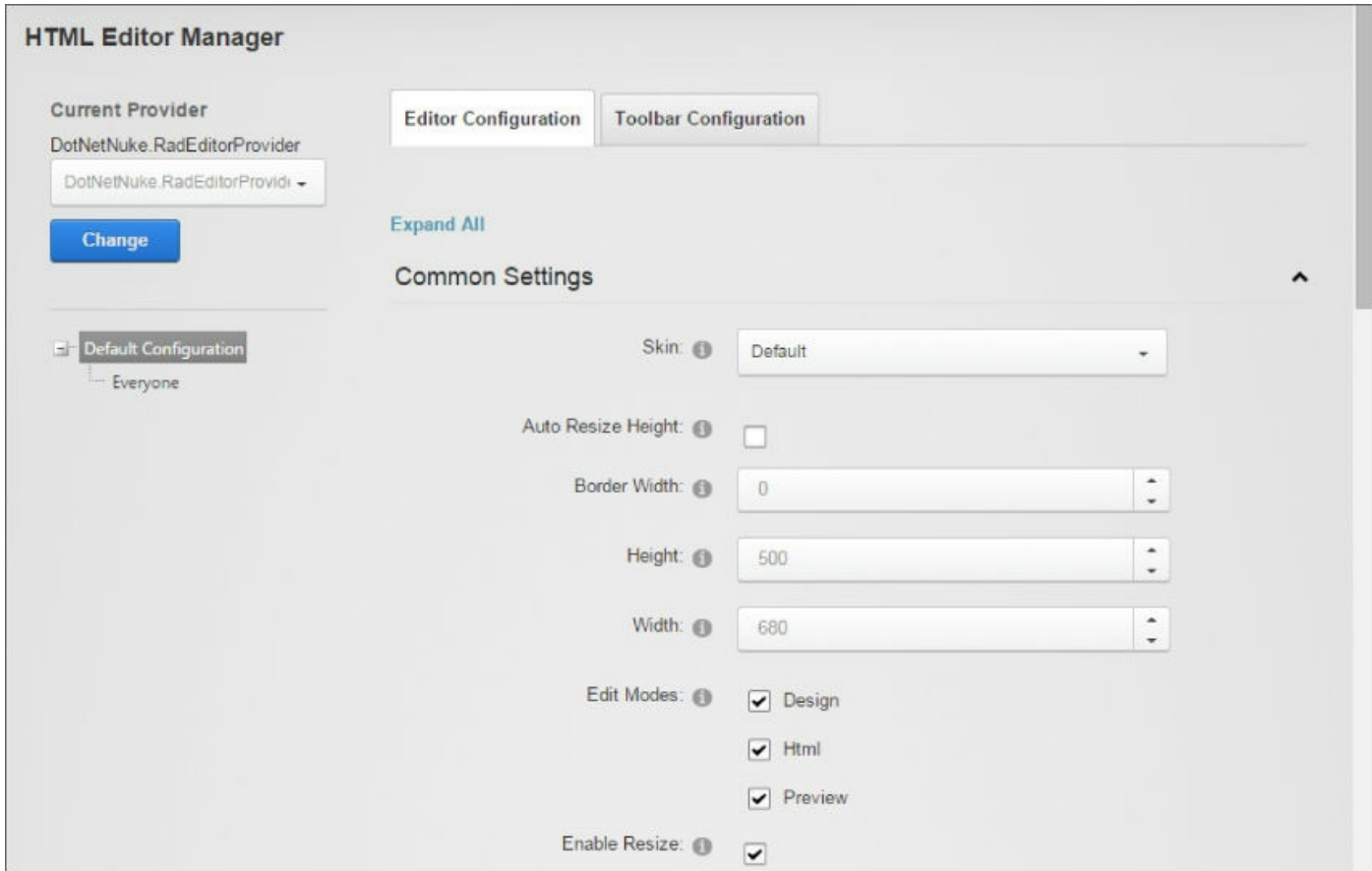


Figure 5.10

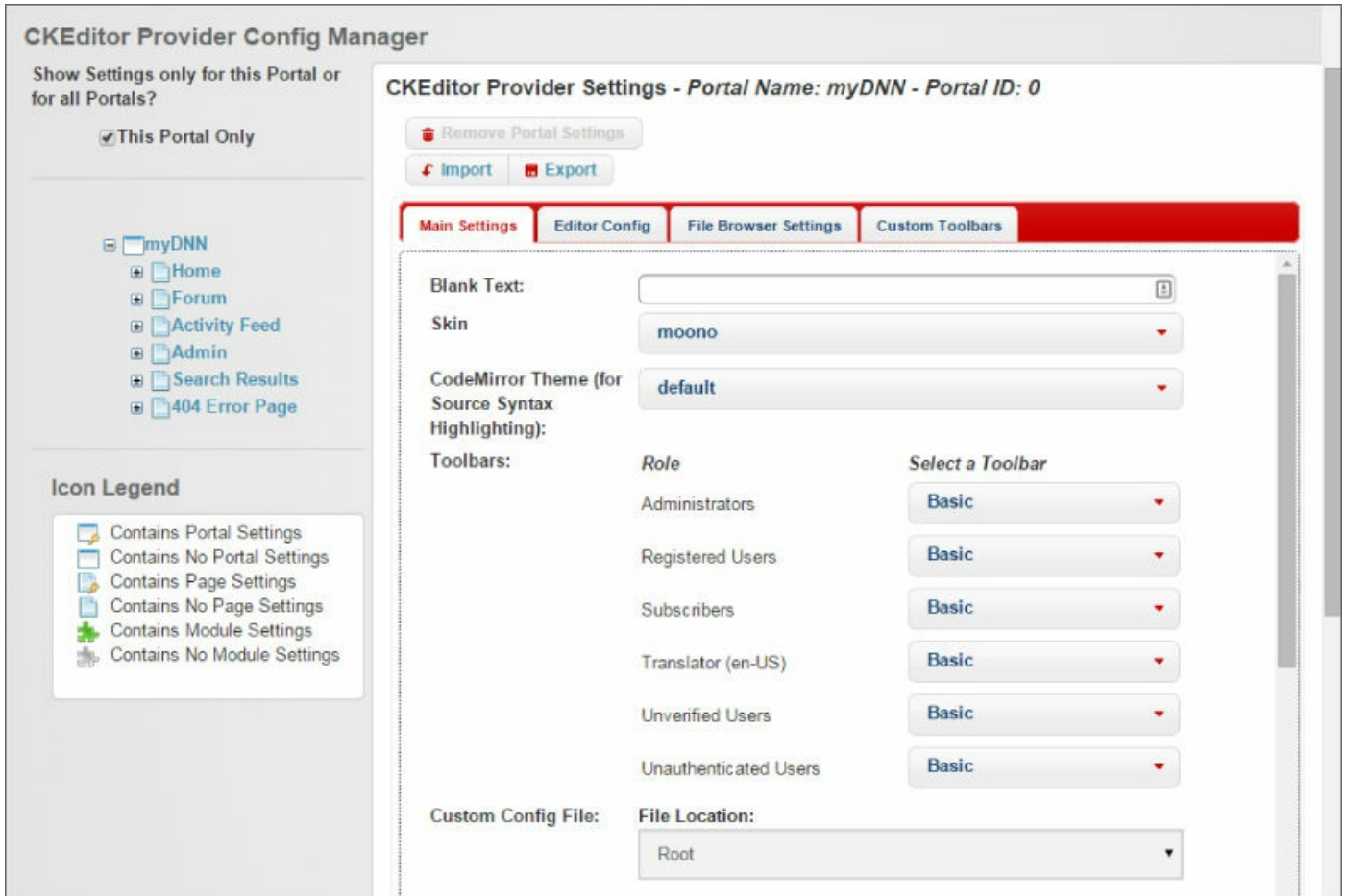


Figure 5.11

Bear in mind that the HTML editor is by far the most utilized feature of any DNN site. All content editors, be they marketing personnel writing page copy, bloggers, or forum posters, use one. Many modules reference the configured HTML provider and so your settings choices will matter a great deal. It is worth spending some time to consider the best settings configurations for your users.

NOTE

The CKEditor provider is an open source project maintained and freely distributed by members of the DNN community (led by Ingo Herbote). You can get it here: <http://dnnckeditor.codeplex.com>.

Lists

Lists were introduced in [Chapter 4](#), as administrators are able to manage Banned Password and Profanity Filter lists specific to their sites. List functionality for hosts is exactly the same, just with different scope. The lists managed by the host include one containing options like Country, State, Image Types, and so on, where lists are used in the UI. User registration, for example, makes use of the Country list.

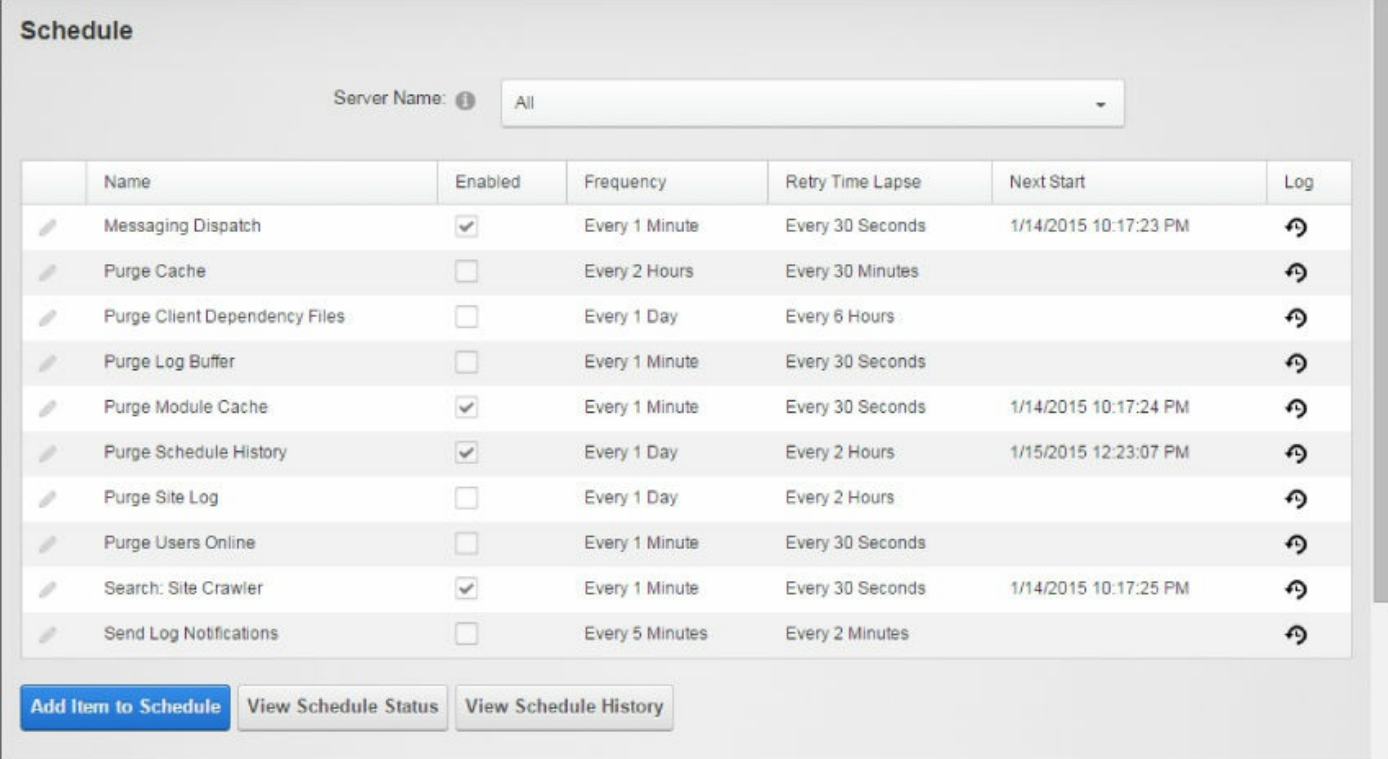
NOTE

When building your own applications, remember the lists functionality built into DNN; there's no need to need to re-create it for your own modules.

Schedule

The Schedule page enables you to manage what is essentially recurring, scheduled batch processing for DNN. Batch processing is a bit of a non sequitur in the typical web environment, but it's actually a very powerful feature in DNN, enabled in large part by the workings of ASP.NET.

[Figure 5.12](#) illustrates a default Schedule page. Notice that some jobs run every minute, whereas others run only once a day.



The screenshot shows the 'Schedule' page in DNN. At the top, there is a 'Server Name' dropdown menu set to 'All'. Below this is a table with the following columns: Name, Enabled, Frequency, Retry Time Lapse, Next Start, and Log. The table contains ten rows of scheduled tasks. At the bottom of the page, there are three buttons: 'Add Item to Schedule', 'View Schedule Status', and 'View Schedule History'.

Name	Enabled	Frequency	Retry Time Lapse	Next Start	Log
Messaging Dispatch	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds	1/14/2015 10:17:23 PM	
Purge Cache	<input type="checkbox"/>	Every 2 Hours	Every 30 Minutes		
Purge Client Dependency Files	<input type="checkbox"/>	Every 1 Day	Every 6 Hours		
Purge Log Buffer	<input type="checkbox"/>	Every 1 Minute	Every 30 Seconds		
Purge Module Cache	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds	1/14/2015 10:17:24 PM	
Purge Schedule History	<input checked="" type="checkbox"/>	Every 1 Day	Every 2 Hours	1/15/2015 12:23:07 PM	
Purge Site Log	<input type="checkbox"/>	Every 1 Day	Every 2 Hours		
Purge Users Online	<input type="checkbox"/>	Every 1 Minute	Every 30 Seconds		
Search: Site Crawler	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds	1/14/2015 10:17:25 PM	
Send Log Notifications	<input type="checkbox"/>	Every 5 Minutes	Every 2 Minutes		

[Figure 5.12](#)

The Scheduler simply initiates processes to run on a host-defined schedule. It supports recurrence, retry, catch-up, event triggers (currently only app restart), dependencies on other scheduled processes so they don't run concurrently, and even specification of which server to run on. This last item

is important in the context of a web farm, as some jobs make sense to run on every server, whereas others must run only once against business information. If two web heads run from the same configuration, it's necessary to ensure that they know which one is responsible for running the job.

The Scheduler can be disabled or set to run in one of two modes: Timer or Request method. These options are tied to an understanding of how “batch processing” is accomplished on a web server. In IIS, a site (or in our case, a DNN instance) is loaded into memory when it receives page requests. The application is kept in memory while it is being used and for some time after, but if a site is quiet for too long, IIS will recycle those resources and remove it from memory. This is why batch processing is not a typical feature in web environments; when the site goes away, so does the batch processor.

DNN provides two ways of contending with this limitation. The Timer method assumes that the site is always loaded into memory, and the Request method relies on the site being “woken” by page requests, whereupon the Scheduler runs. Each has its limitations, but the preferred method is Timer, which turns your attention to the means necessary to keep a site “awake.”

All it takes to keep a site awake is for traffic to occur before IIS recycles the ASP.NET worker process. A busy global site likely has enough traffic to keep it loaded in memory, but for many sites this can't be counted on. However, there are many services that can be used to periodically “ping” a site. In fact, these are quite common now and often part of a comprehensive site health-monitoring approach.



NOTE

Consider the use of a service like Pingdom, CopperEgg, Uptime Robot, or StatusCake to provide site-monitoring and “keep alive” services for your site. Each of these vendors has free offerings.

Additionally, the Delay Schedule at Start setting can be managed on this page. When a site is recycled, a lot of work goes on when it starts to “warm up” again (building the cache and such). This setting will cause the Scheduler to wait the specified number of minutes before triggering its first job on app start.

Schedule Item Details

[Figure 5.13](#) illustrates the detail page for a Scheduler entry, namely, the Purge Schedule History job. You may recall that one of the settings in Host Settings was the number of days of Schedule History to maintain. This job's function is to trim the Schedule History each day to conform to those settings.

myDNN > Schedule > Edit Schedule

Friendly Name:

Full Class Name and Assembly:

Schedule Enabled:

Schedule Start Date/Time:

Frequency: Days

Retry Time Lapse: Hours

Retain Schedule History:

Run on Event:

Catch Up Enabled:

Object Dependencies:

Run on Servers:

Figure 5.13

Let's take a quick look at some of these fields.

For the developers, the Full Class Name and Assembly option refers to an assembly in the `<Root>/bin` directory that inherits from `DotNetNuke.Services.Scheduling.SchedulerClient`. Any extension could include and set up a scheduled job as part of its installation script, if desired. The Start Date/Time is optional, and if it's not supplied, the job timer will simply start from the creation of the Scheduler entry.

NOTE

Developers should note that the Scheduler is a host service, running outside of `HttpContext`. That means `PortalSettings` are not available. If site context is needed, it must be garnered in an alternative way.

The Object Dependencies field permits specification of one or more string values that are meant to serve as semaphores between jobs to avoid potential deadlock conditions. Consider, for example, two jobs that both update the `Folders` table. By placing the same arbitrary string value on both jobs, say “FileLock,” the Scheduler will not permit both to run at the same time. It will delay the second job until the first has completed.

Configuration

The Scheduler also has a couple of settings in the `web.config` file that are useful for development and/or troubleshooting, as follows:

- `debug`—When this is set to true, the Scheduler becomes verbose, generating a substantial amount of `EventLog` table entries. Debugging multithreaded applications is challenging, and this can be very helpful in development of scheduler jobs.
- `maxThreads`—Any number greater than -1 will enforce a maximum number of threads that can be utilized by the Scheduler. A value of -1 will leave thread management to the Scheduler itself (which currently causes it to use a single thread).

SQL

The SQL console permits database access directly from inside DNN. So the host, even if it's not near an instance of SQL Server Management Studio, still has access to a powerful database tool. Files containing SQL commands can be imported from disk and also saved in DNN for later use. [Figures 5.14](#) and [5.15](#) illustrate the SQL console editor and the tabbed query results with paging.

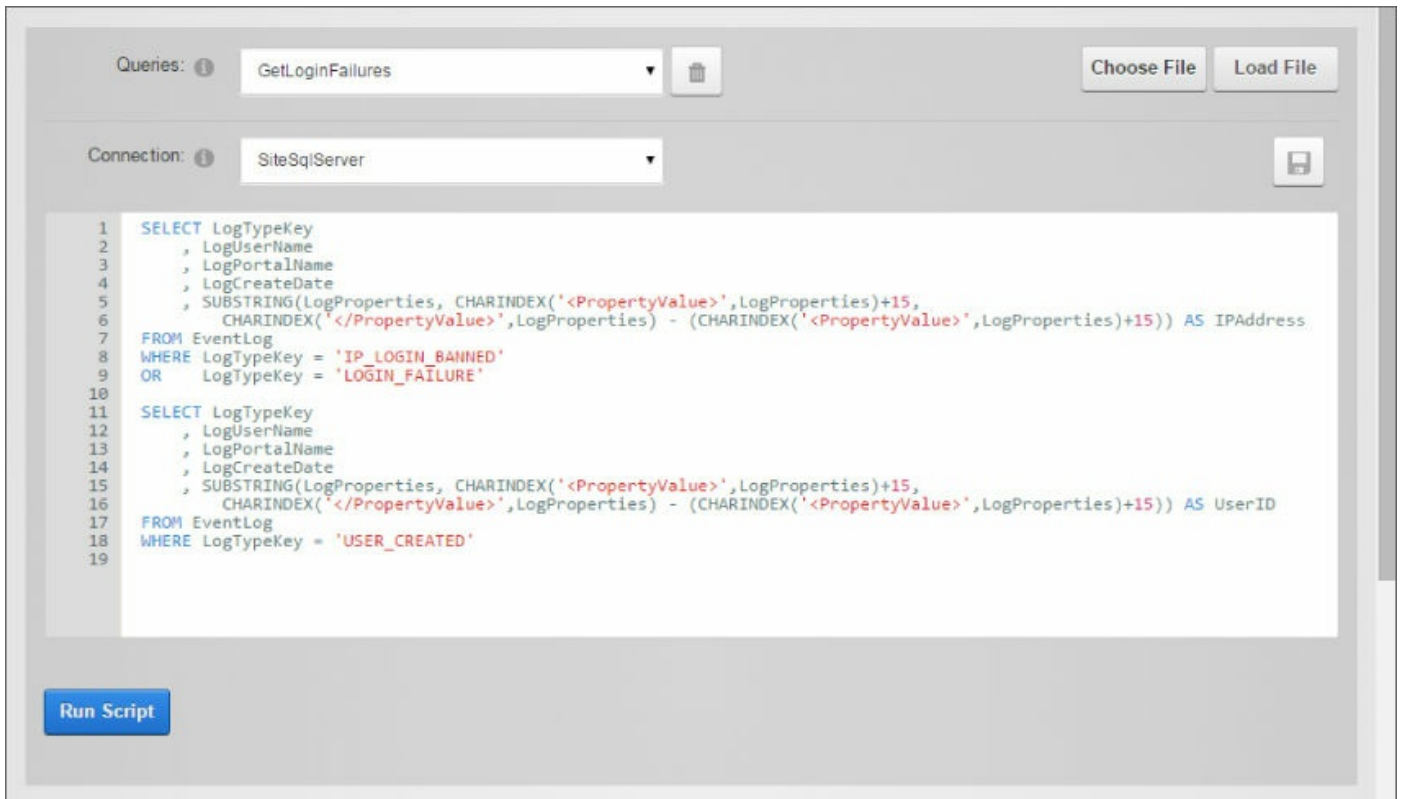


Figure 5.14

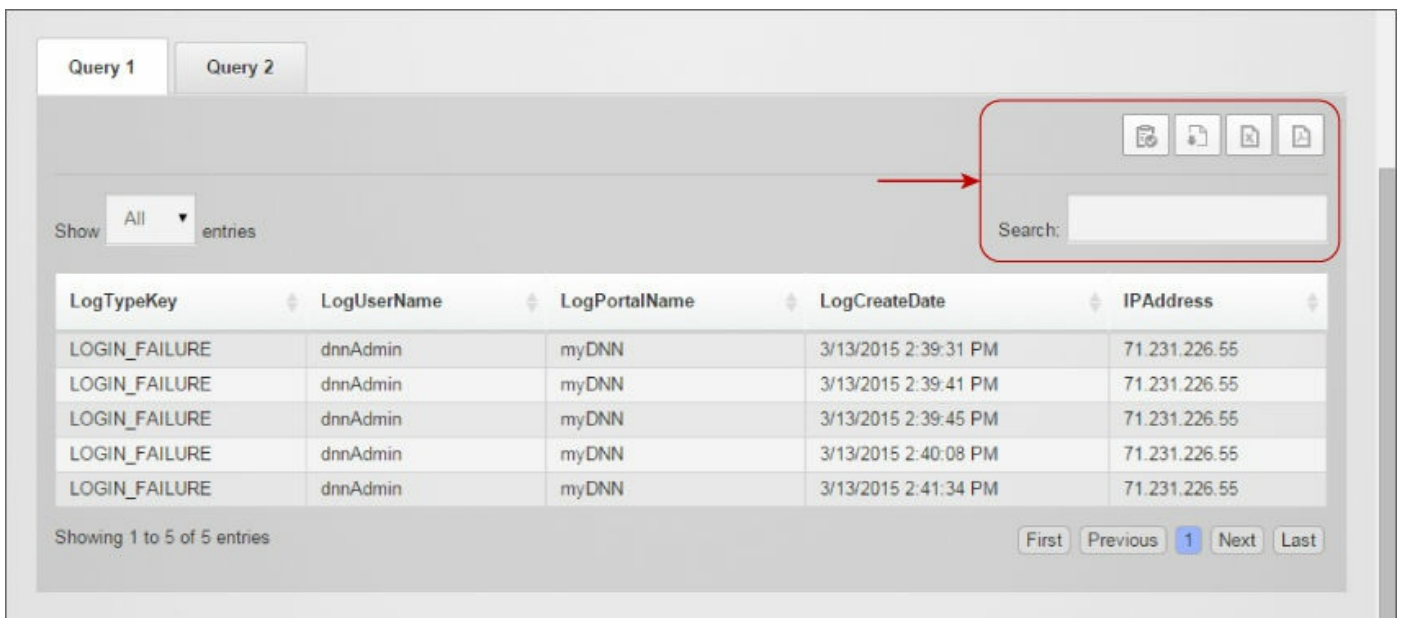



Figure 5.15

Note the ability to search a large returned dataset and buttons that support copying results to the clipboard and exporting results to CSV, Excel, or PDF formats.

Vendors

You were introduced to the concept of *vendors* and their association with banner advertising in [Chapter 4](#). Host vendors are simply a collection of vendors that belong to the host rather than to a specific site. When implementing a banner module, you may specify whether it is associated with host or site vendor listings.

Additional Host Features on Admin Site Settings

In addition to his own menu of features, the host also has access to site-specific settings. As mentioned earlier, these settings can be reached via the Host  Site Management feature, but it's generally better to navigate directly to them in the context of the site you are updating. The Site Switch option under the Tools menu will provide a quick means to switch between sites.

So let's look again at the Admin menu you were introduced to in [Chapter 4](#), where you will now find all the same pages but several new categories and settings within them.

Event Viewer

You learned about the Event Viewer in [Chapter 4](#), even about the additional functionality available to the host: clearing the log, changing log settings, and email notification on selected event types. One additional host feature simply provides the host a view of the Event Log entries by site. This view can be particularly helpful when investigating the source of errant behavior in order to note whether errors are coming predominantly from one site or another.

Site Settings

The majority of new categories are found on Site Settings. Let's explore these.

Advanced: Site Aliases

Site aliases tell DNN what domain names a site should respond to. You will recall that DNN supports the concept of virtualized sites within a single instance. This means that multiple sites, each potentially with multiple unique URLs, can exist in one instance of DNN, that is, one set of files and one database. For DNN to know what site a request should load, it uses a system of site aliases. When a page request is received from IIS, it extracts the domain name portion, compares against the list of site aliases, and then redirects to the relevant site to load the appropriate page.

NOTE

Developers should note that it's very helpful to work with a site with a legitimate domain name rather than simply `localhost` or an IP address, especially when trying to debug or investigate issues in a staging environment. Domain names can be easily “spoofed” on your local machine with a simple entry in the

`c:\windows\system32\drivers\etc\hosts` file (note that the `hosts` file has no file extension and requires Windows Administrator privileges to edit). The following entry will enable the `dnndev.me` domain to be treated, on your machine, as if it were registered in DNS.

```
127.0.0.1 dnndev.me
```

[Figure 5.16](#) illustrates a local install with two aliases for the same site in the same domain. The primary address has the form of a “child site” (a subdirectory beneath a fully qualified domain name—FQDN), while the other is its own FQDN.

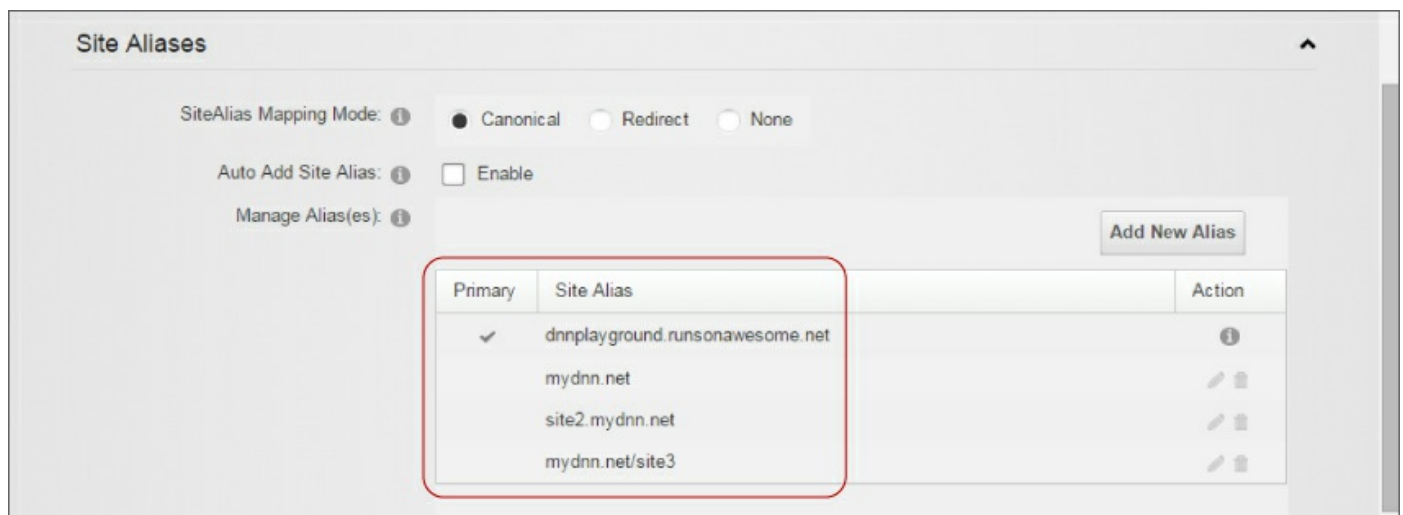
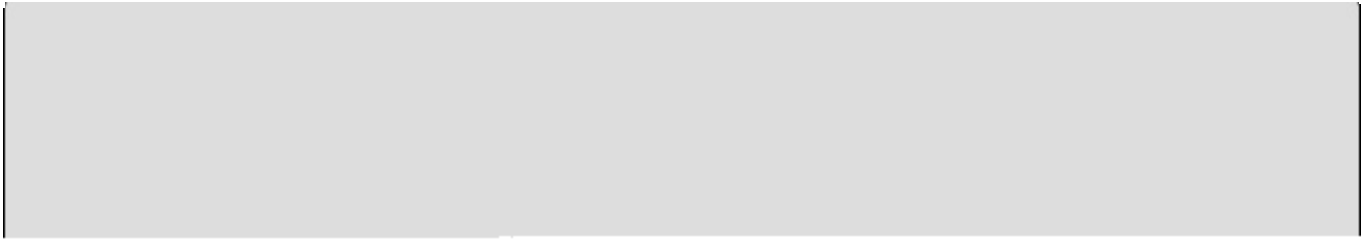


Figure 5.16

It is not uncommon for work on new sites to begin under a `localhost` or a locally established FQDN (like `dnndev.me`), even with a “child” configuration. A child site can be “promoted” simply adding an FQDN alias, ensuring IIS is properly configured to direct those requests to the DNN instance, and removing the physical directory that was created for the child alias (such as `<Root>\site2`). Don't worry, there's nothing in the directory that can't be simply deleted.



NOTE

Note that use of aliases in multiple domains for the same site can have significant impacts on cookie handling, SEO, and SSL certificates. Be sure to develop your domain strategy and review it for potential pitfalls before you get started.

In a final word on site aliases, we recommend that your default choice of Mapping Mode always be Redirect. Search engines like Google don't like duplicate content, and pages with duplicate content don't rank well.

DNN site pages are dynamically generated, and there is a risk of generating duplicate content where the same page is served with a variety of URLs. Take these cases, for example:

- dnndev.me
- www.dnndev.me/default.aspx
- dnndev.me/home.aspx
- www.dnndev.me/home/tabid/56/default.aspx

Each of these URLs might easily resolve to the same content—the home page of a DNN site. Compound this problem with duplicates generated from a third-party module (like a blog or forum), and you could have hundreds or even thousands of duplicate URLs on a DNN site.

You can set the site alias Mapping Mode to establish either primary or canonical domain behavior. Primary behavior (the “Redirect” option) means that when a non-primary alias is received, a 301 redirect is issued to the primary site alias. The implication is that over time, search engines will start recognizing that all those entry points to your site represent one set of content and stop splitting your search ranking points among the various URLs.

Canonical behavior means that requests to valid site aliases will be respected (without redirection), but a Canonical Link element will be generated in the page's HTML header. In general, the “Redirect” is preferred, but you might have a particular strategy for leveraging multiple aliases to show the same content, in which case the “Canonical” option will work better.

Advanced: SMTP Server Settings

We went over SMTP Server Settings earlier in the chapter, and there's no reason to go over them again here. Just recall that the host settings determine the default for new site creation, whereas selecting “portal” (or “site”) mode in the site settings allows for site-specific SMTP server settings. In this way, each site can be configured to use the appropriate SMTP server for its domain's mail services.



NOTE

Sharing SMTP services among multiple domains is a good way to get your SMTP server blacklisted, ensuring your mail always winds up in your users' SPAM folders.

On somewhat of a side note, if you're doing a lot of work on outbound mail, say configuring email layouts, generating outbound messages, and so on, you might not want to use an actual SMTP server. PaperCut, an open source community project, is a simplified SMTP server designed for exactly this purpose. It permits you to intercept “sent” mail and examine its contents without it leaving your machine. You can get PaperCut at

<https://papercut.codeplex.com>.

Advanced: SSL Settings

These days, sites that don't implement SSL, at least for login and registration, are frowned upon. But you can configure DNN to utilize a site-specific or shared SSL certificate.



NOTE

You can test SSL behavior by creating your own local certificate to use in IIS. It won't have a certifying authority, of course (so you will get a warning), but it will behave correctly. The `MakeCert.exe` (Certificate Creation) tool is included in the .NET Framework SDK.

The SSL Enabled checkbox tells DNN that when pages are specified as “secure” in their page settings that it can utilize the HTTPS protocol with the additional settings to follow. Note that when this setting is not enabled, the Secure option is not available.

Without the SSL Enforced option enabled, pages that are not marked as “secure” cannot be accessed under HTTPS. You may want to turn this on if you're using a specific URL for HTTPS traffic that you don't want users to see on other pages (such as secure.example.com).

The SSL URL is typically used only in a shared hosting scenario where individual sites cannot implement their own certificates. This URL would be provided by the hosting provider; but you should also ensure it is added to the site as a legitimate site alias.

The standard URL is required when using the alternate SSL URL. It specifies a return page (on your site), where users will be directed when they move from secure to unsecure pages.

NOTE

Don't lock yourself out! It is quite possible to set up SSL incorrectly on your first try, resulting in some odd behavior and even the potential to lock yourself out of your site by making login pages inaccessible. If this happens, you might need to remove the “secure” page settings directly in the database to recover. The following SQL will reset all the pages in a specific site, removing any “secure” settings (clear the cache or restart the website after running this script):

```
UPDATE TabsSET IsSecure = 0WHERE IsSecure = 1AND PortalID = <your  
site id #>
```

Advanced: Messaging Settings

Messaging Settings govern the internal sending of messages by site users. [Figure 5.17](#) illustrates a User Profile page with one “message” in the inbox. These settings are used primarily to guard against potential abuses of the messaging system by spammers or otherwise malicious users.

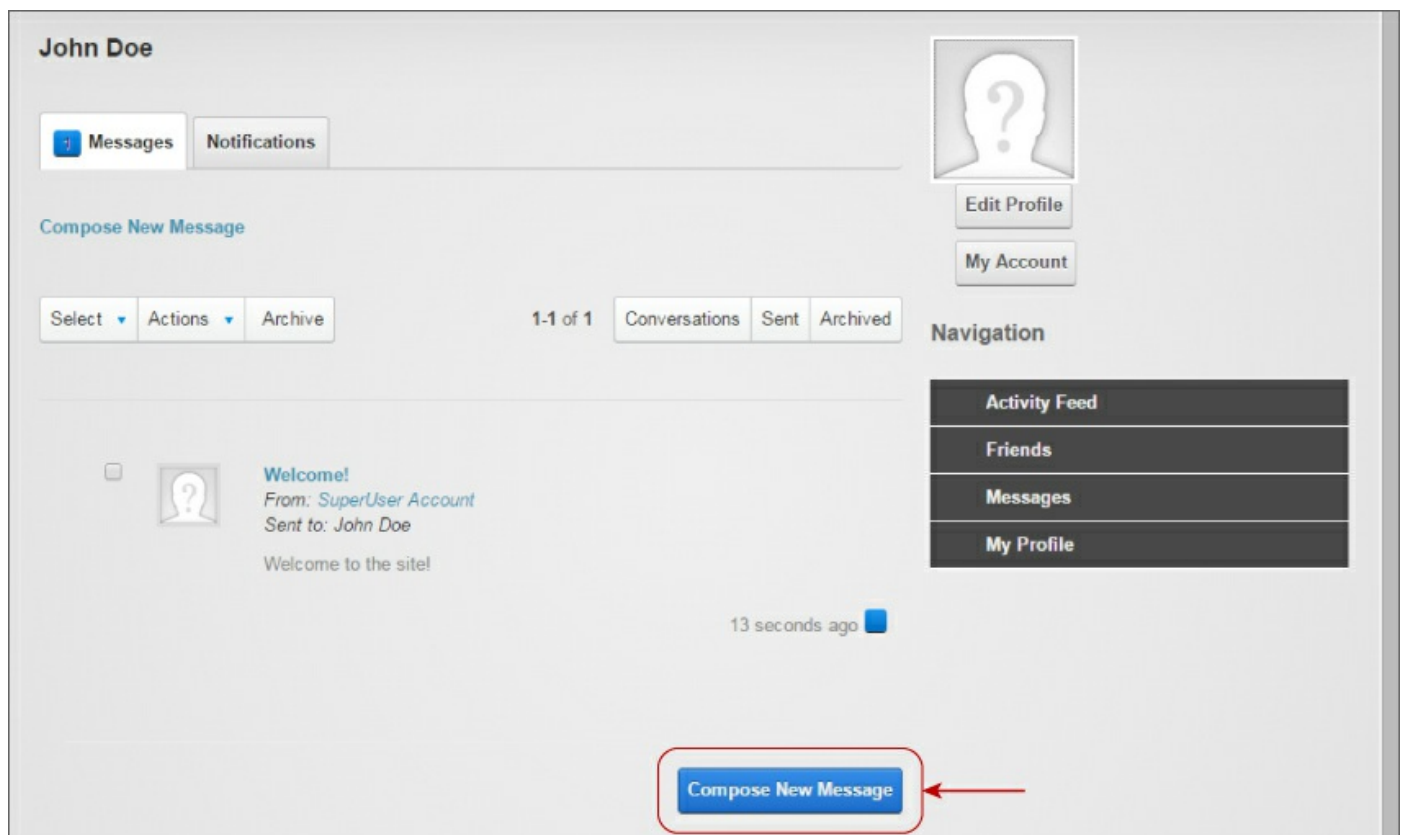
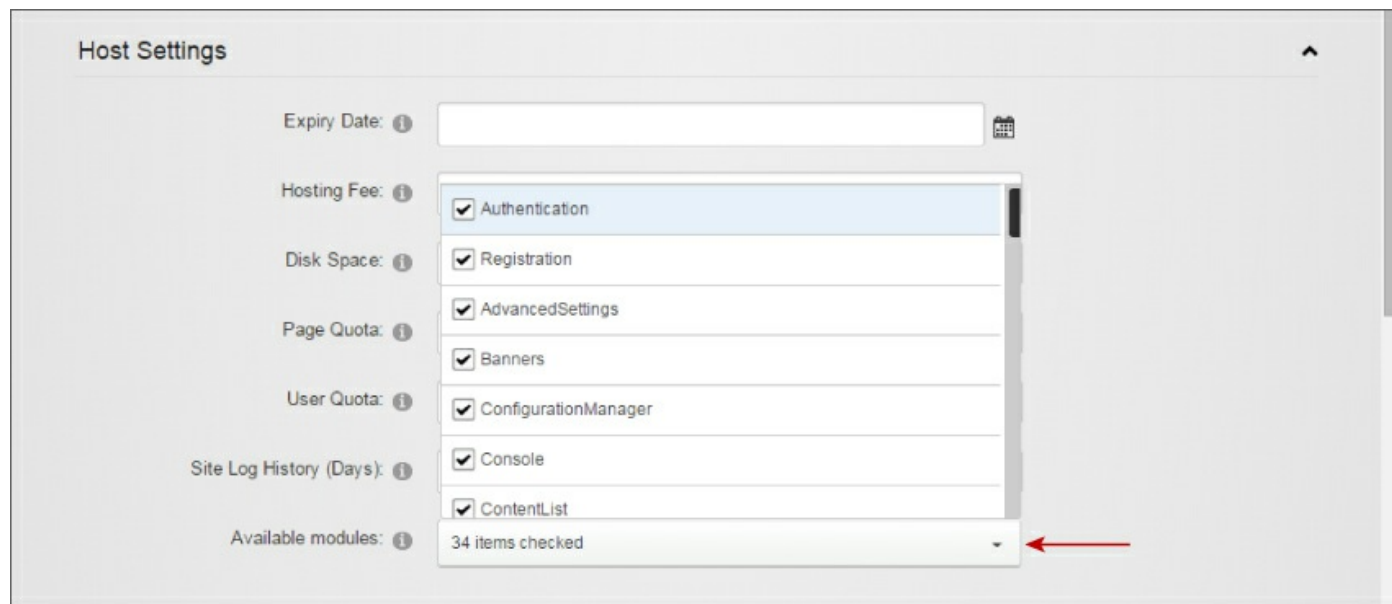


Figure 5.17

These options are pretty straightforward, and the help is comprehensive.

Advanced: Host Settings

The Host Settings were discussed earlier in the chapter, but there is one item in the Host Settings that merits pointing out, as shown in [Figure 5.18](#).



[Figure 5.18](#)

The Available Modules selector indicates which installed modules are accessible for the site administrators to deploy on their site. You'll notice in [Figure 5.18](#) that there are “35 items checked” and some of those items don't really look appropriate to be deployed without some oversight (such as `AdvancedSettings`, `CKEditor.EditorConfigManager`, `ConfigurationManager`, and so on).

By default, installed modules are all available for new site admins to deploy and use, even admin modules! You might want to change this. Simply uncheck items from this list that you don't want site administrators to be able to deploy without consulting the host.

Page Management

When logged in as the host, Page Management now includes a selector to distinguish between host and site pages; otherwise, it functions in the same way as explained in [Chapter 4](#). See [Figure 5.19](#) for the host user view.

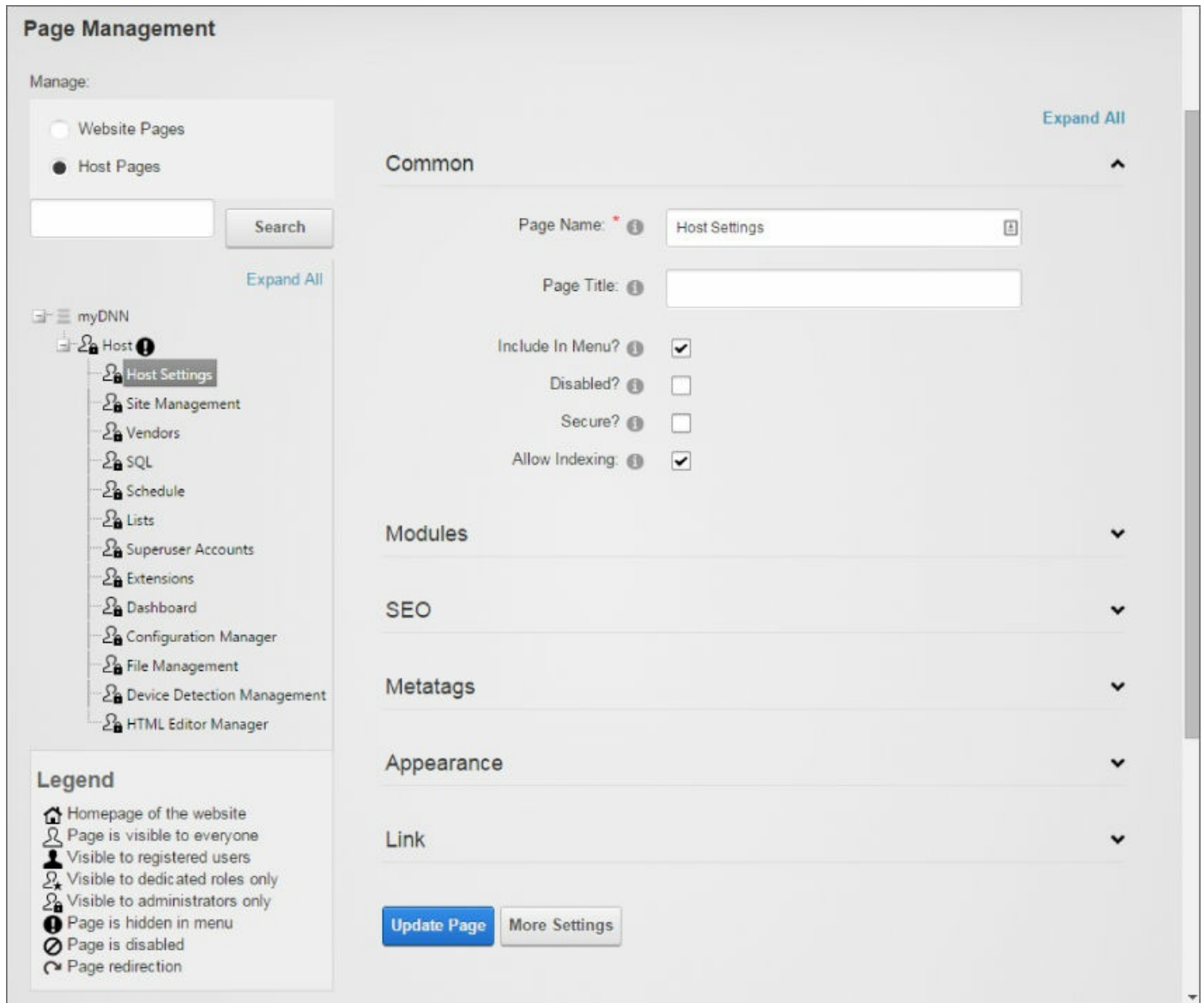


Figure 5.19

Advanced Configuration Settings

There are a number of additional tabs on this page that are visible to the host. However, they are all simply alternative methods of accessing functionality that is discussed elsewhere. Specifically, the following additional tabs are visible:

- **SMTP Server:** As discussed in Host Settings ↗ SMTP Server Settings
- **Language Packs:** As discussed in Host ↗ Extensions ↗ Available Extensions, although limited to the Language Packs category alone
- **Authentication Systems:** As discussed in Host ↗ Extensions ↗ Available

Extensions, although limited to the Authentication Providers category alone

- **Providers:** As discussed in Host ⇨ Extensions ⇨ Available Extensions, although limited to the Providers category alone
- **Optional Modules:** As discussed in Host ⇨ Extensions ⇨ Available Extensions, although limited to the Modules category alone

Languages

You may recall that [Figure 5.9](#) illustrated the effects of enabling the Enable Localized Content checkbox. [Figure 5.20](#) identifies the additional features available to the host from the Language Management page.

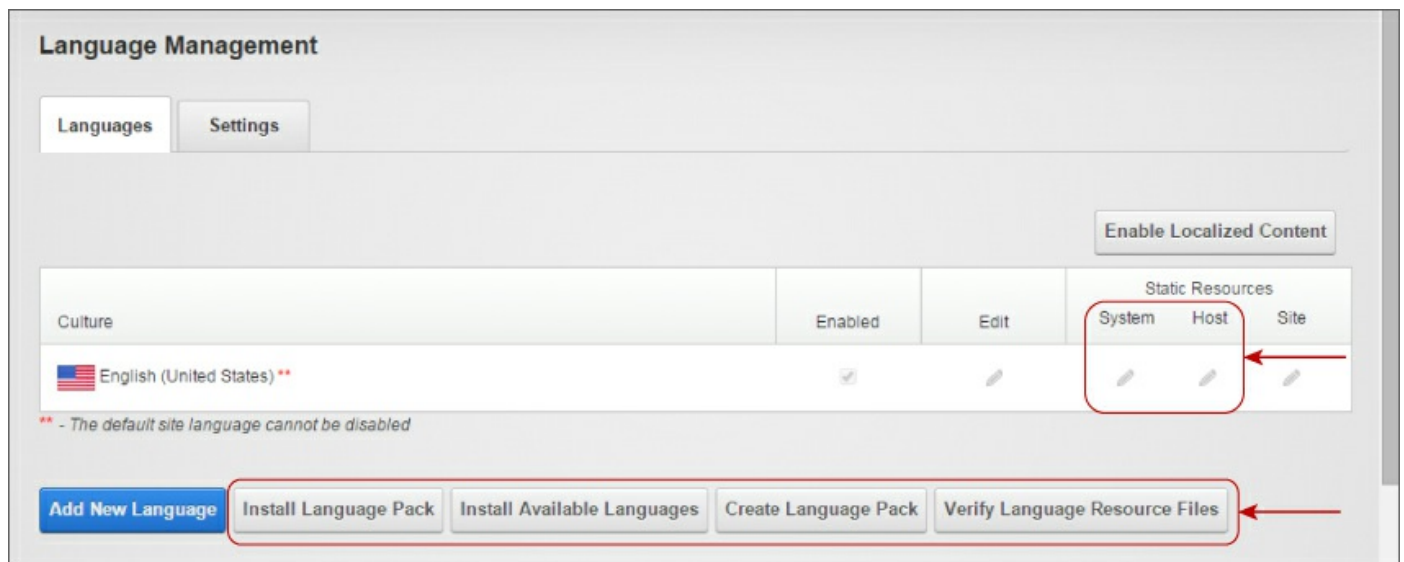


Figure 5.20

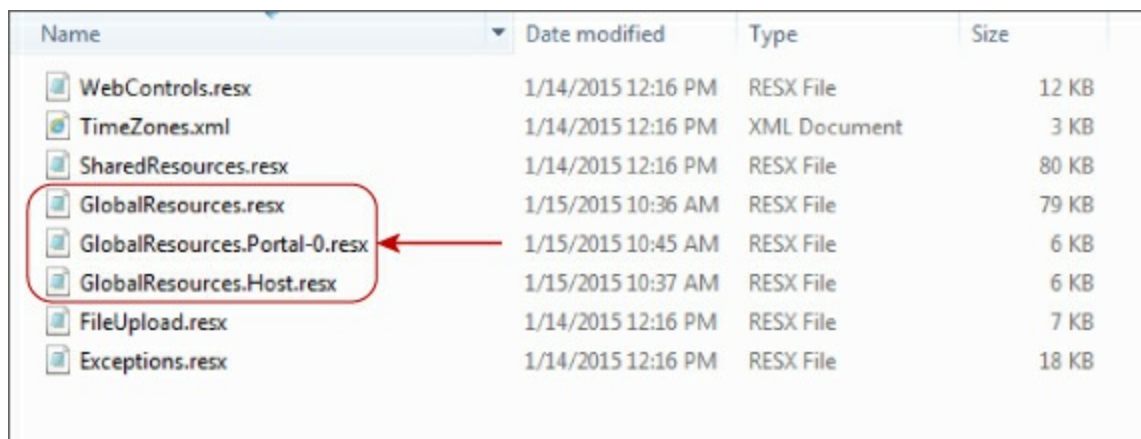
We'll leave the buttons for discussion in [Chapter 10](#). But let's have a closer look at the new static resources available to the host for edit: System and Host.

Resource files utilize a fallback structure, which means that only changes from base files are required for customization. DNN searches first in Site resources, then in Host resources, and finally in System resources to locate the strings it will use. The system file is considered the root file and should remain unchanged unless you're planning to customize an install procedure. Changes to the system resource files essentially cannot be undone because the original values are overwritten. This differs from the host and site resource files, which are constructed as deltas from the system- and host-

level resources files, respectively.

Let's say, for example, that I change the password reminder email (resource name: `EMAIL_PASSWORD_REMINDER_BODY.Text`) in the host resource file. This creates a new default inherited by every site in the DNN instance. Where the original file in `<Root>\App_GlobalResources` was named `GlobalResources.resx`, the new host-level file is named `GlobalResources.Host.resx`. It does not contain all the entries from the system file, only the changes made to personalize it.

Then, a site admin decides she would like to make the password reminder even more personalized for the site, so she changes the resource, which produces a new file named `GlobalResources.Portal-0.resx`. The site ID is appended to `Portal-` to distinguish it from resource files customized for other sites in the instance. A quick look at the `<Root>/App_GlobalResources` folder in [Figure 5.21](#) illustrates the result.



Name	Date modified	Type	Size
WebControls.resx	1/14/2015 12:16 PM	RESX File	12 KB
TimeZones.xml	1/14/2015 12:16 PM	XML Document	3 KB
SharedResources.resx	1/14/2015 12:16 PM	RESX File	80 KB
GlobalResources.resx	1/15/2015 10:36 AM	RESX File	79 KB
GlobalResources.Portal-0.resx	1/15/2015 10:45 AM	RESX File	6 KB
GlobalResources.Host.resx	1/15/2015 10:37 AM	RESX File	6 KB
FileUpload.resx	1/14/2015 12:16 PM	RESX File	7 KB
Exceptions.resx	1/14/2015 12:16 PM	RESX File	18 KB

[Figure 5.21](#)

Additional Host Features on the Control Panel

So far we've examined an entire menu structure dedicated specifically to host features. And we've re-examined the site administrators feature menu to see what additional features have been revealed for the host. Now, we turn our attention to the remaining menus of the control panel.

Tools Menu

A couple of additional items appear on the Tools menu when you are logged in as host. The Clear Cache and Recycle Application Domain buttons were discussed previously under the Host Settings topic. These shortcuts perform in exactly the same way and are placed here as a convenience. Recall the caveats of utilizing these two commands in a production environment!

A handy additional feature on the Tools menu is the Site Switcher (see [Figure 5.22](#)). This feature allows the host to quickly change context.

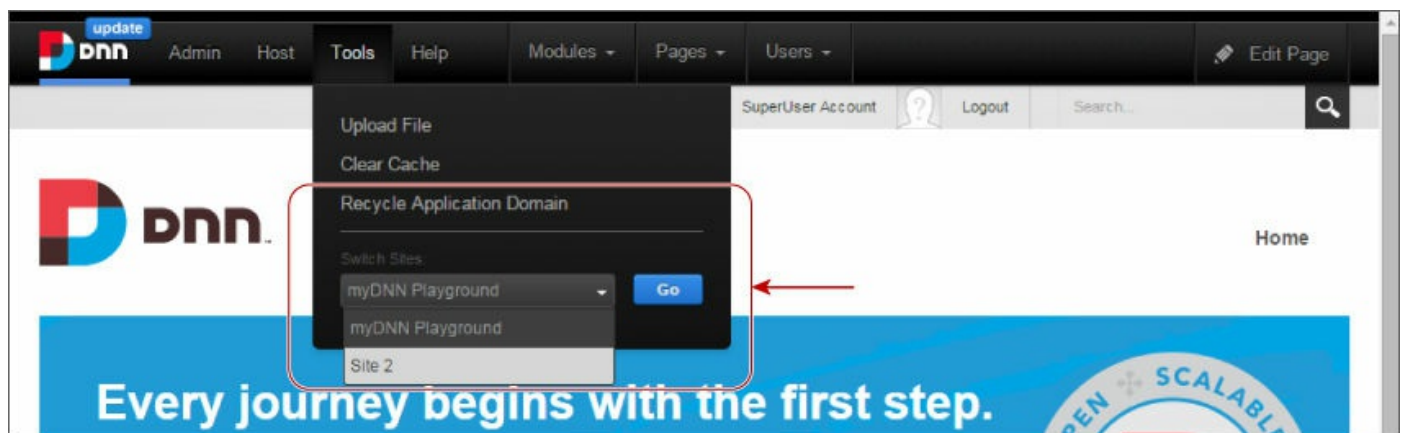




Figure 5.22

In order to make site-specific changes, a host can utilize the Host  Manage Sites page and update settings in that manner. Alternatively, a host can simply navigate to that site, sign in, and use the Site Settings pages of that specific site. The Site Switcher simply makes it easy to navigate between sites, switching to their primary alias by clicking Go.

Modules Menu

Two new items are added to the Modules menu for the host. The first item, called Find More Modules, is simply a direct shortcut to the Host  Extensions page featuring the More Extensions tab.

The second item is an additional shortcut called Create Module, and it's a single-step operation that puts the current page into Edit mode and opens the module list to the developer category, by default featuring the Module Creator. This is a handy shortcut for adding developer components to a page without going through too many steps.

Host Options on the Module Actions Menu

Finally, the last host-only feature appears on the Module Actions menu. This menu is associated with every module instance, as illustrated in [Figure 5.23](#). The new option on this menu is called Develop.

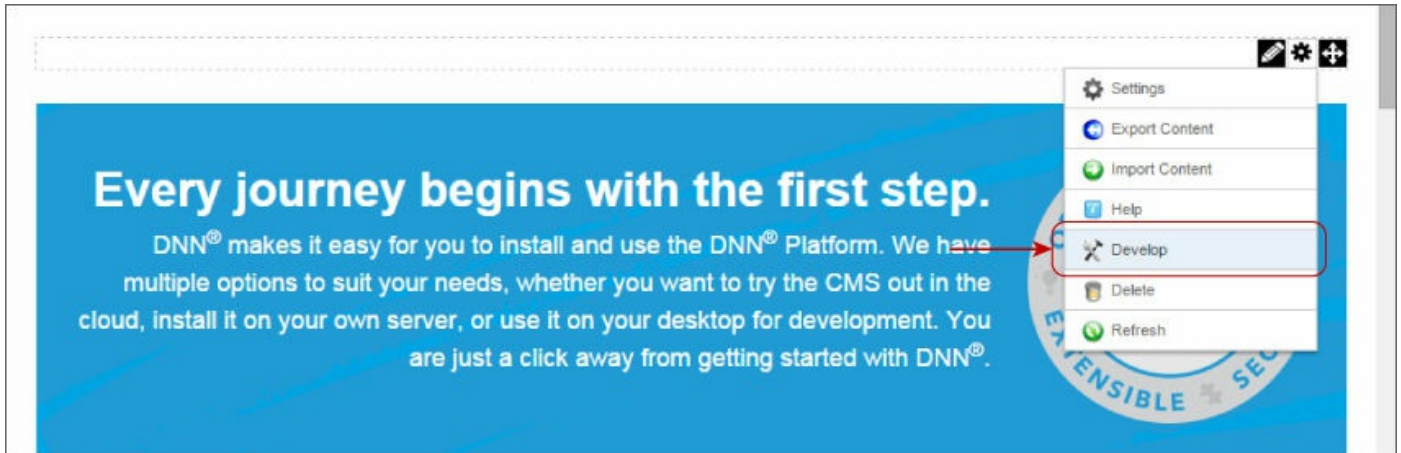


Figure 5.23

If you're not a developer, this feature may not be of interest to you, but if you are, you may wind up referencing it extensively. Selecting the Develop action results in loading of the module's ASCX file into an editor, where it can be modified on the fly as necessary. Other files associated with the module can also be opened and edited, including the code-behind (if it's present in a source code deployment).

This is very powerful capability and strong support for developers, making it entirely possible to build and package complete applications from within DNN. This is discussed in [Chapter 13](#). The View Source editor is illustrated in [Figure 5.24](#).

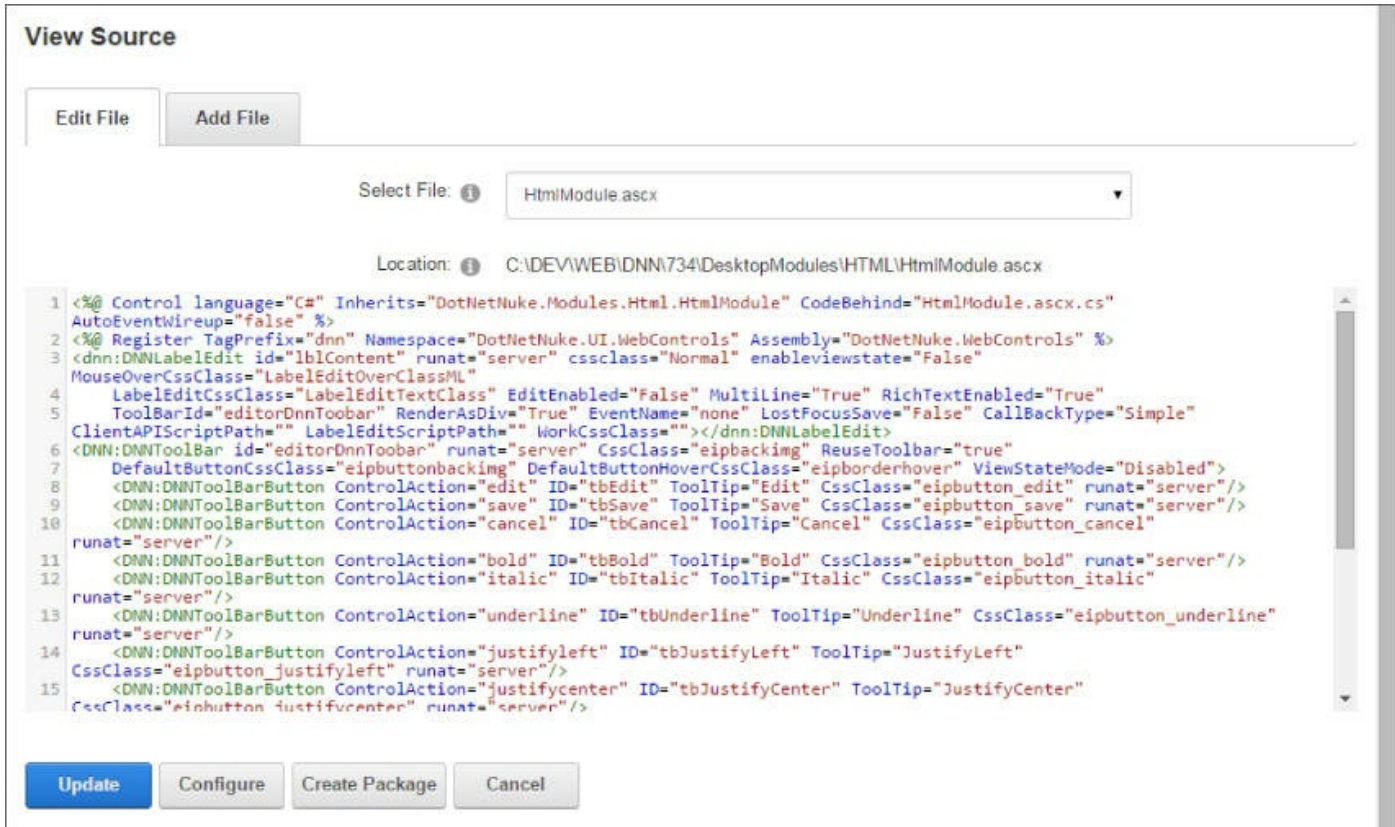


Figure 5.24

Integrating with a Third-Party Provider

Well, now that you've learned where all the interesting bits are, it's time to put some of them to work. In the next couple of pages, you'll learn how the host can extend the DNN instance by installing third-party modules and integrating with external services such as Facebook, Google, and Twitter.

Technically, the only part of this operation that cannot be performed by a site administrator is installing the providers, but for that reason this kind of configuration usually falls to a host to complete.

Installing the Facebook Provider

Authenticating site access with third parties like Facebook, Google, or Twitter has become almost an expectation on the Web today, and DNN includes authentication providers to support each of these. Navigate to the Host Extensions page and then scroll down and expand the Authentication Systems section. As you can see in [Figure 5.25](#), the Facebook Provider is ready to be installed!

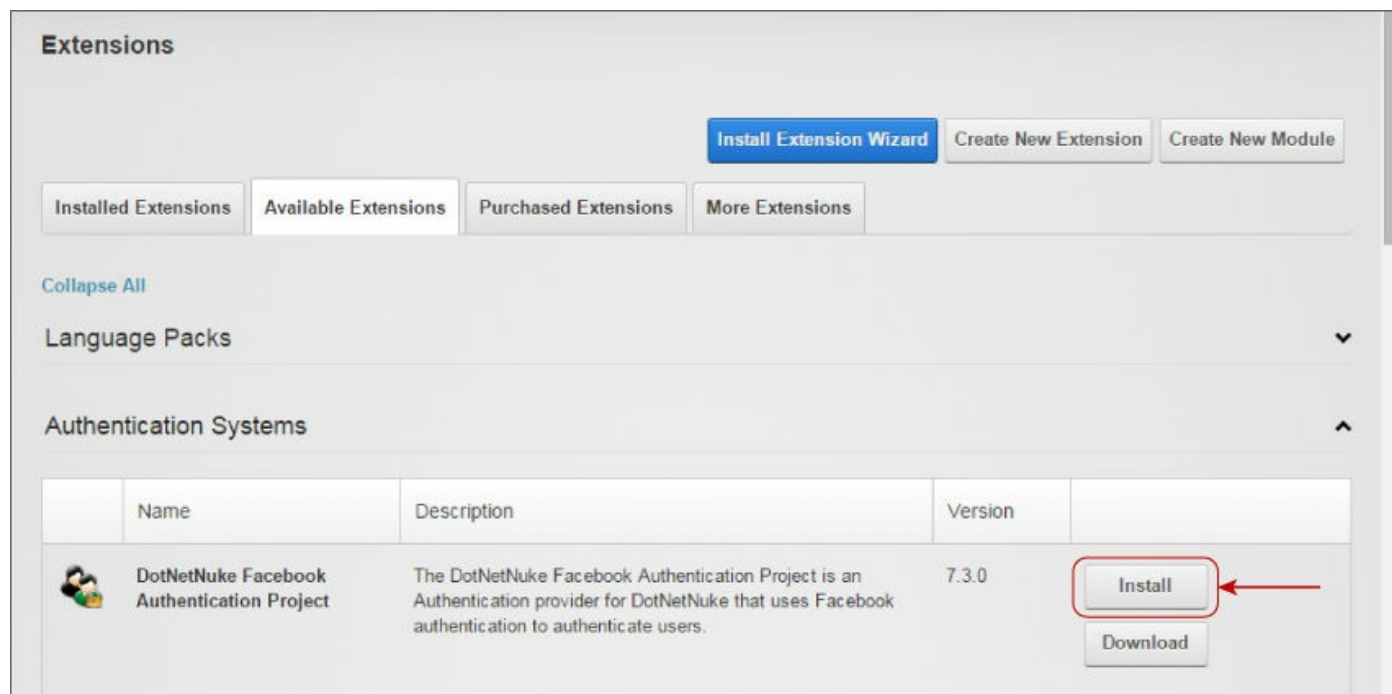


Figure 5.25

Clicking Install, you can proceed through the Install wizard. You can view details and release notes and accept the license, as illustrated in [Figure 5.26](#). [Figure 5.27](#) shows the successful Package Installation Report.

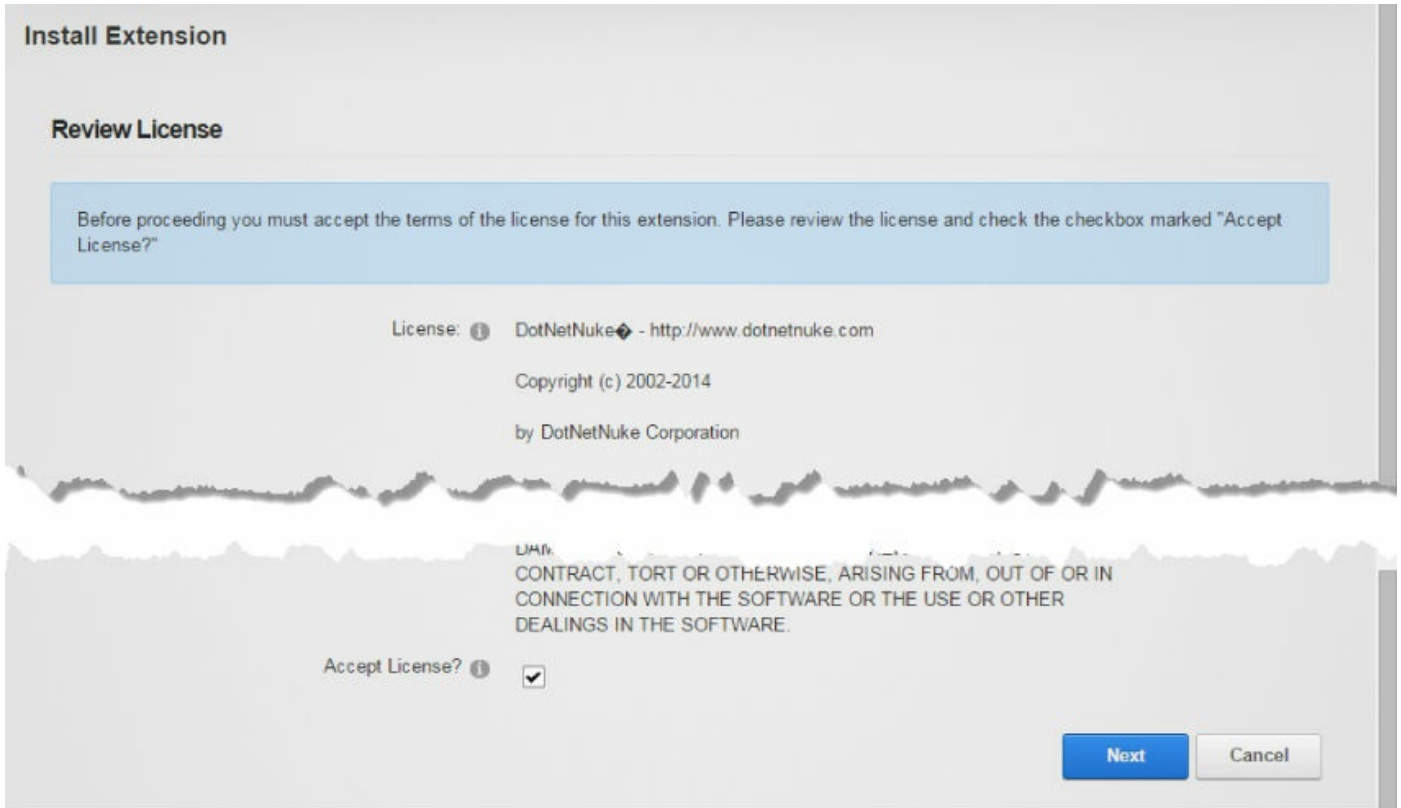


Figure 5.26

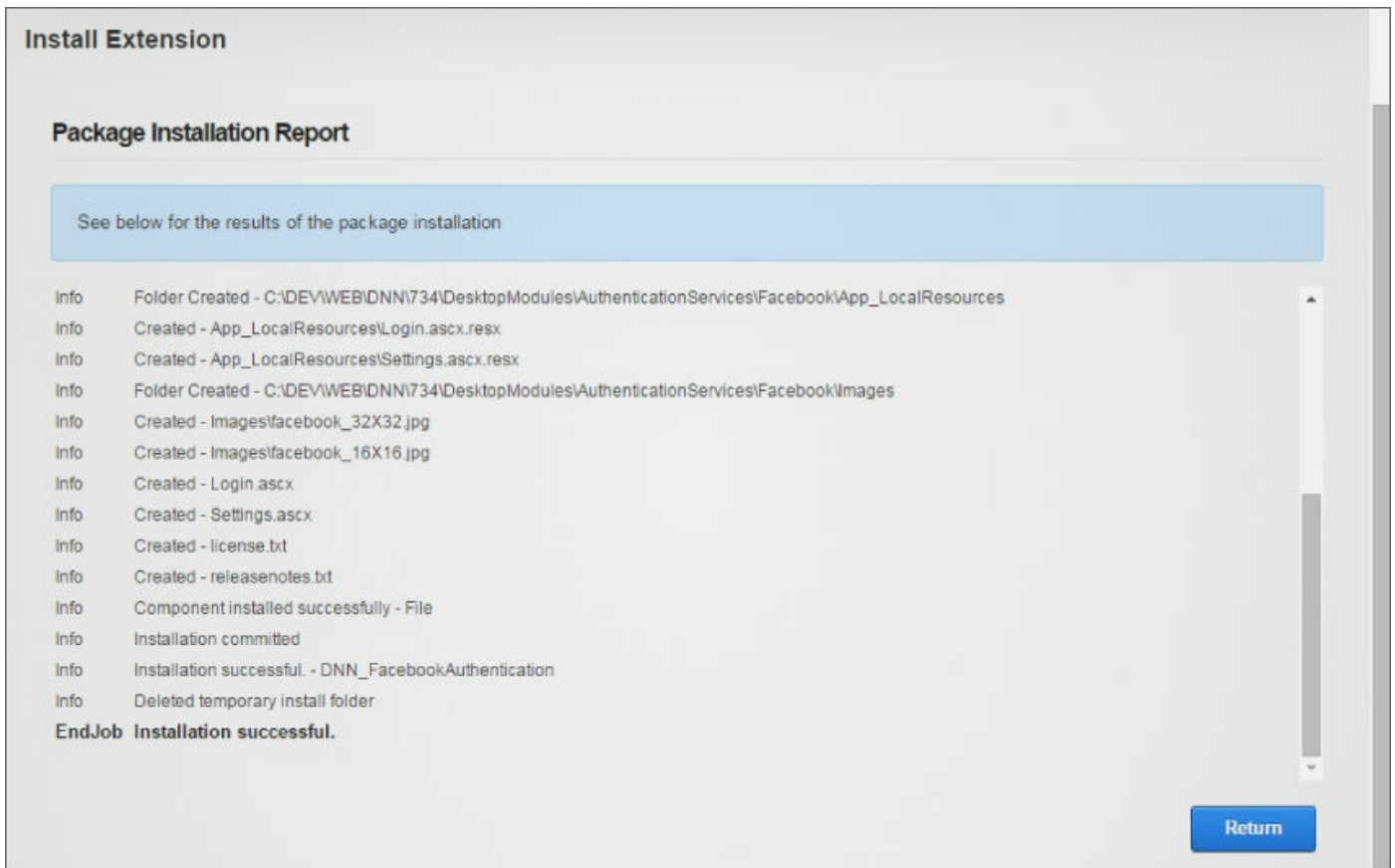


Figure 5.27

When you return to the list of available authentication providers, note that Facebook is no longer present. That's because it is now located on the Installed Extensions tab.

Setting Up the Facebook Application

Setting up the DNN Facebook Provider requires a Facebook application to connect to. Despite how it sounds, this does not require any coding. You simply fill out some information in the developer area of Facebook, which defines a new application to generate the keys required for your DNN instance to “talk” to Facebook.

Navigate to <https://developers.facebook.com> and log in. From the My Apps menu, select Add a New App, as illustrated in [Figure 5.28](#). When you're prompted for your platform, look at the bottom of the dialog and select Advanced Setup. This will take you straight to the Create a New App ID window. Simply give the ID a friendly name and select a category, as shown in [Figure 5.29](#). Finally, click Create App ID.



NOTE

Be advised that Facebook likes changing its interfaces frequently, so your experience may vary. But in the end, you just need to acquire a new app ID.

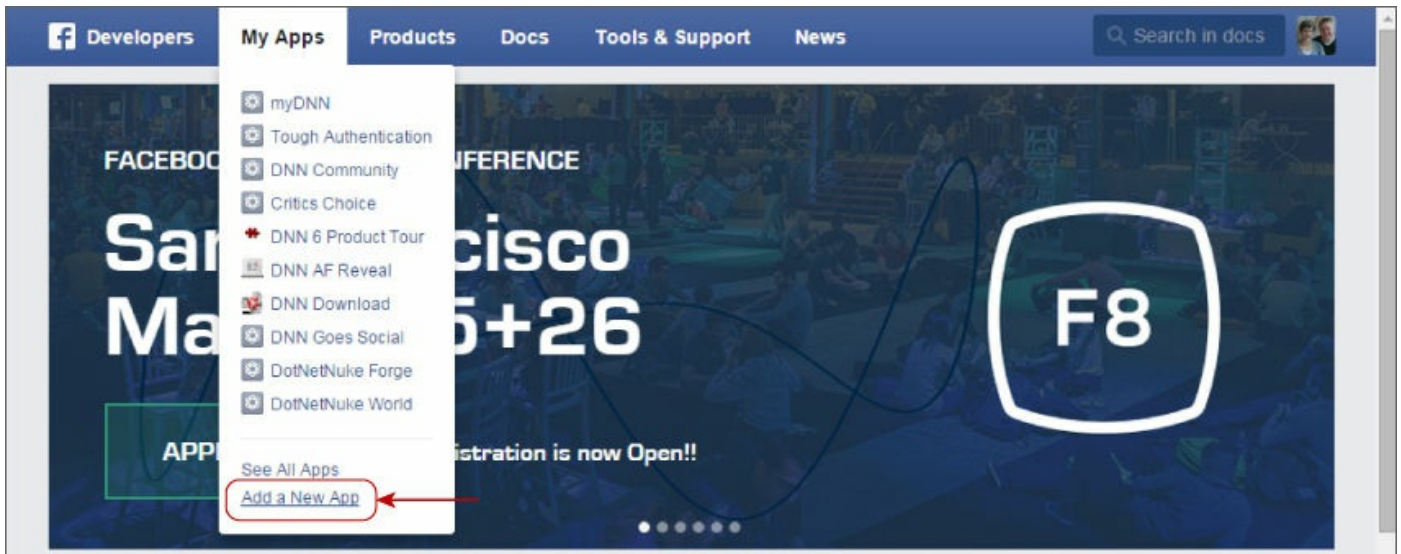


Figure 5.28

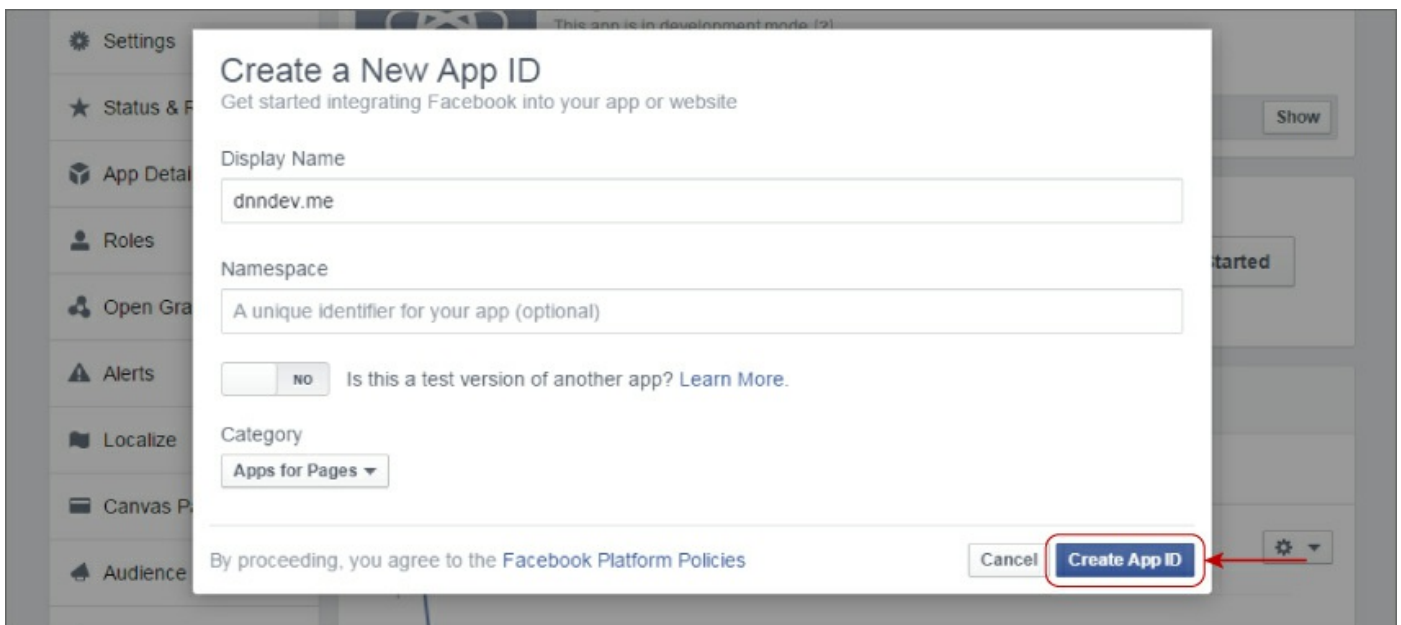


Figure 5.29

After clicking the Create App ID button, you should return to the Dashboard of your new Facebook App, complete with its own App ID and App Secret (see


[Figure 5.30](#)). You need these keys to configure DNN in the next step.

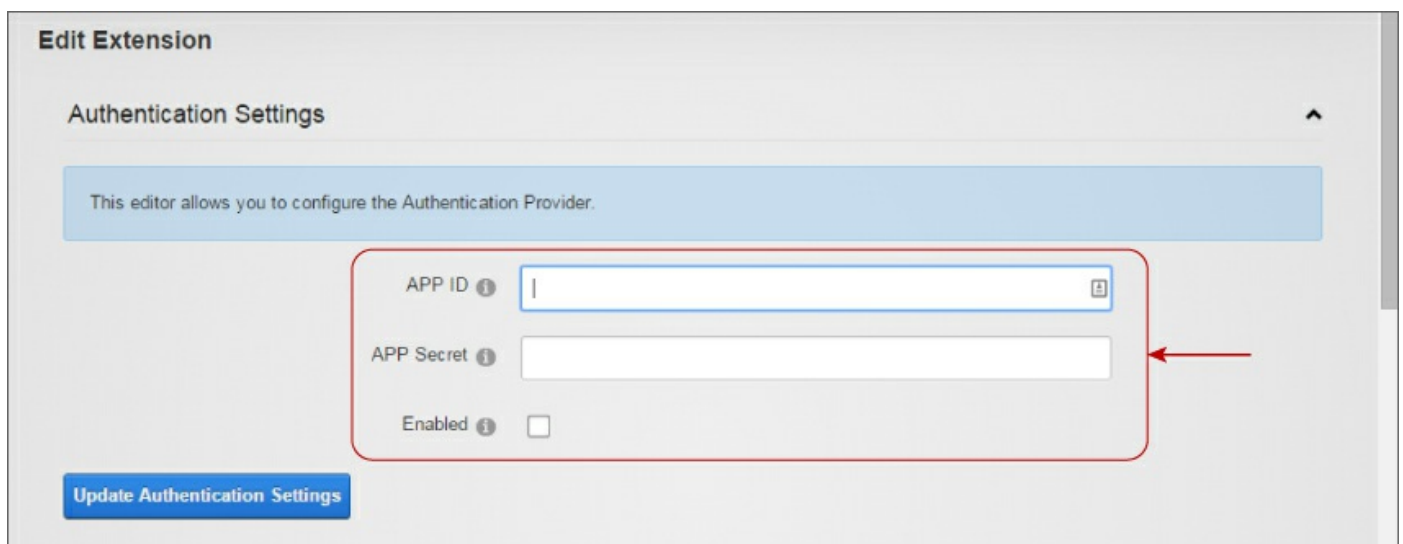


[Figure 5.30](#)

Activating the Facebook Provider

Now that you have an installed provider and a viable Facebook application, you're ready to activate the provider for use on your site!

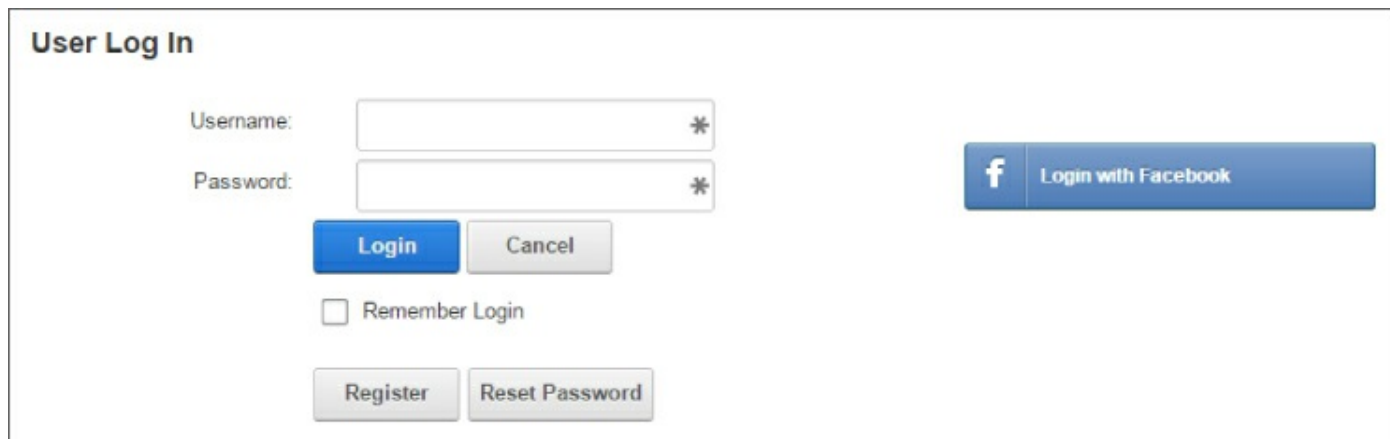
Back in your DNN instance, you'll want to navigate to the Admin  Extensions page of the site you want to support Facebook authentication. In the Authentication Providers section, click the pencil icon on the far right so that the Facebook Authentication Provider opens its settings. See [Figure 5.31](#).



[Figure 5.31](#)

Add the App ID and App Secret you just created and check the Enabled

checkbox. Finally, click the Update Authentication Settings button. On your next login attempt, you'll be greeted with a slightly different page, as illustrated in [Figure 5.32](#)!



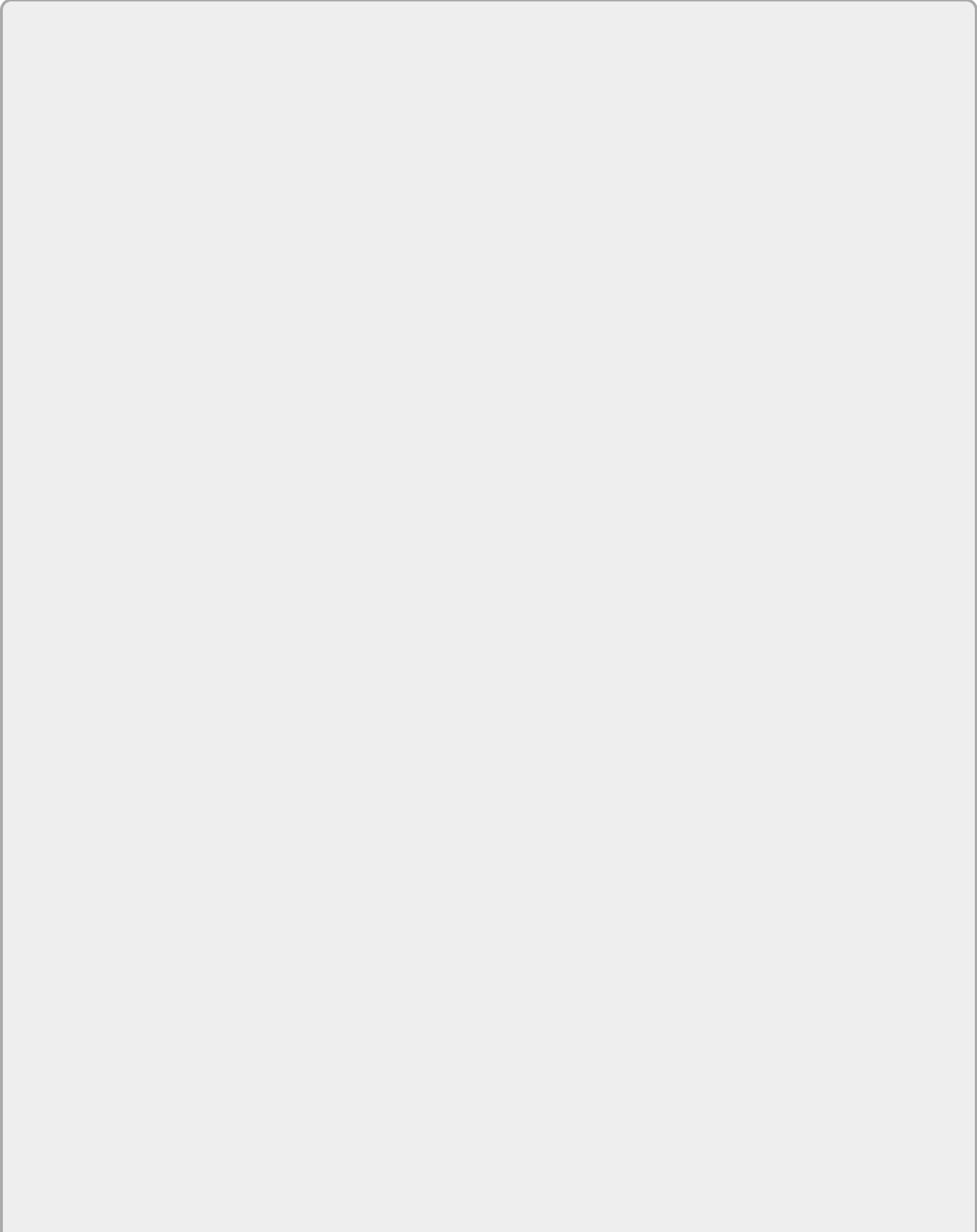
The image shows a 'User Log In' form. It features two input fields for 'Username' and 'Password', each with an asterisk on the right. Below these fields are two buttons: 'Login' (blue) and 'Cancel' (grey). A checkbox labeled 'Remember Login' is positioned below the buttons. At the bottom of the form are two more buttons: 'Register' and 'Reset Password'. To the right of the form is a blue button with a white Facebook 'f' icon and the text 'Login with Facebook'.

[Figure 5.32](#)

Summary

Well, congratulations! In this chapter you've learned just about everything there is to know about managing a DNN instance. As you have observed, the host user really is a superuser, with all the privileges of the site administrator you learned about in [Chapter 4](#) as well as a slew of additional features. In many cases the host has access to advanced features on admin pages not visible to the site admin, as well as completely new functionality via the Host menu options. DNN is capable of handling the simplest to the most complex of site designs, and—with a little practice—you'll be able to configure, monitor, troubleshoot, and optimize them all.

Chapter 6
Modules



What You Will Learn In This Chapter

- Adding modules to pages
- Sharing modules across multiple pages
- Configuring module permissions
- Scheduling module visibility by date and time
- Installing modules
- Configuring the HTML rich-text editor

DNN can be extended in several ways. The areas where DNN can be extended are known as extension points. DNN has several extension points such as authentication providers, skins/themes, and language packs just to name a few. Probably the most popular extension point, however, is the one known as “modules.” DNN is built on a modular architecture, and as such, developers can leverage the platform architecture to create reusable, modular units of functionality. This chapter provides a high-level overview of modules. If you are looking for more in-depth information and guides for creating your own custom module, check out [Chapters 13](#) through [16](#).

What Is a Module?

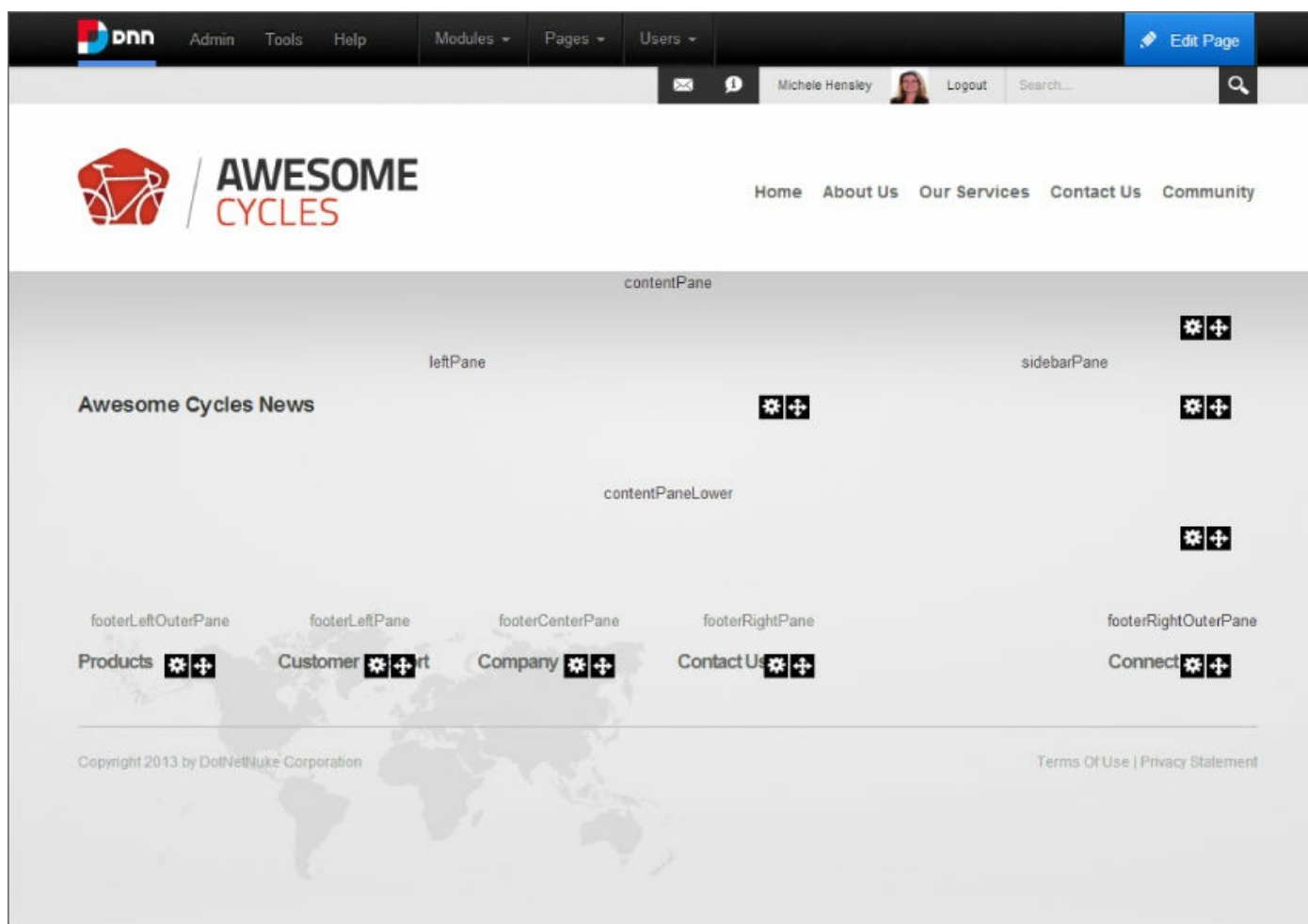
A module is a reusable, plug-and-play piece of functionality. We often use the analogy of a module being to DNN what an app is to an iPhone. It's a way to extend the functionality of the framework with whatever functionality you desire. Examples of modules could be photo galleries, blogs, rotators, forms, and so on. There can be multiple modules on a page and even multiple instances of the same module on a page. It's very common to see multiple HTML modules on a single page.

From a developer's perspective, a module is generally comprised of a set of user controls along with the associated files, scripts, and images needed to carry out the module's functionality. ASP.NET developers are familiar with user controls and commonly use them when creating bits of code that can be reused. For this reason, the learning curve and entry into module development for ASP.NET developers is short and easy.

When you install DNN, the platform ships with a base set of modules already installed in the system. There are too many that come with the solution out of the box to list here, but suffice it to say that all modules needed for basic functionality within a website come with the solution.

Where Do Modules Live on a Page?

Modules are placed onto pages in locations called “panes.” These panes are defined by the skin/theme's designer. The “skin” dictates the overall look and feel of your site and is usually created by a front-end designer. Panes can span the full width of the page or be positioned in a columnar fashion. It may be helpful to think of panes as windows in the design of the site where you can drop in modules. If a pane does not have a module placed in it, then the pane simply does not display when the page is loaded. [Figure 6.1](#) shows a skin's panes. This view is attainable by hovering over the Edit Page menu and clicking View in Layout Mode.



[Figure 6.1](#)

Adding a Module to a Page

In order to add a module to a page, simply hover over the Modules menu in the Control Panel at the top of the page and then click Add New Module. A drop-down menu that spans the width of the screen appears. This menu shows all available modules that exist in your site. Your cursor defaults to the module search bar where you can easily search for modules by typing in the first few letters of your module's name. This list can also be filtered by category by clicking the Category drop-down list just to the left of the search box.

The categories by which you can filter modules exist in the system by default, and host users can add additional categories if they so choose. A small horizontal scroll bar is present at the bottom of the module drop-down menu that users can click and slide to the right and left to quickly find the module for which they are looking. Hovering your mouse over any of the modules turns the module blue, and a pop-up message is displayed indicating that you can drag and drop the module onto the page. [Figure 6.2](#) shows the Modules drop-down menu that a user sees when adding a module to a page.

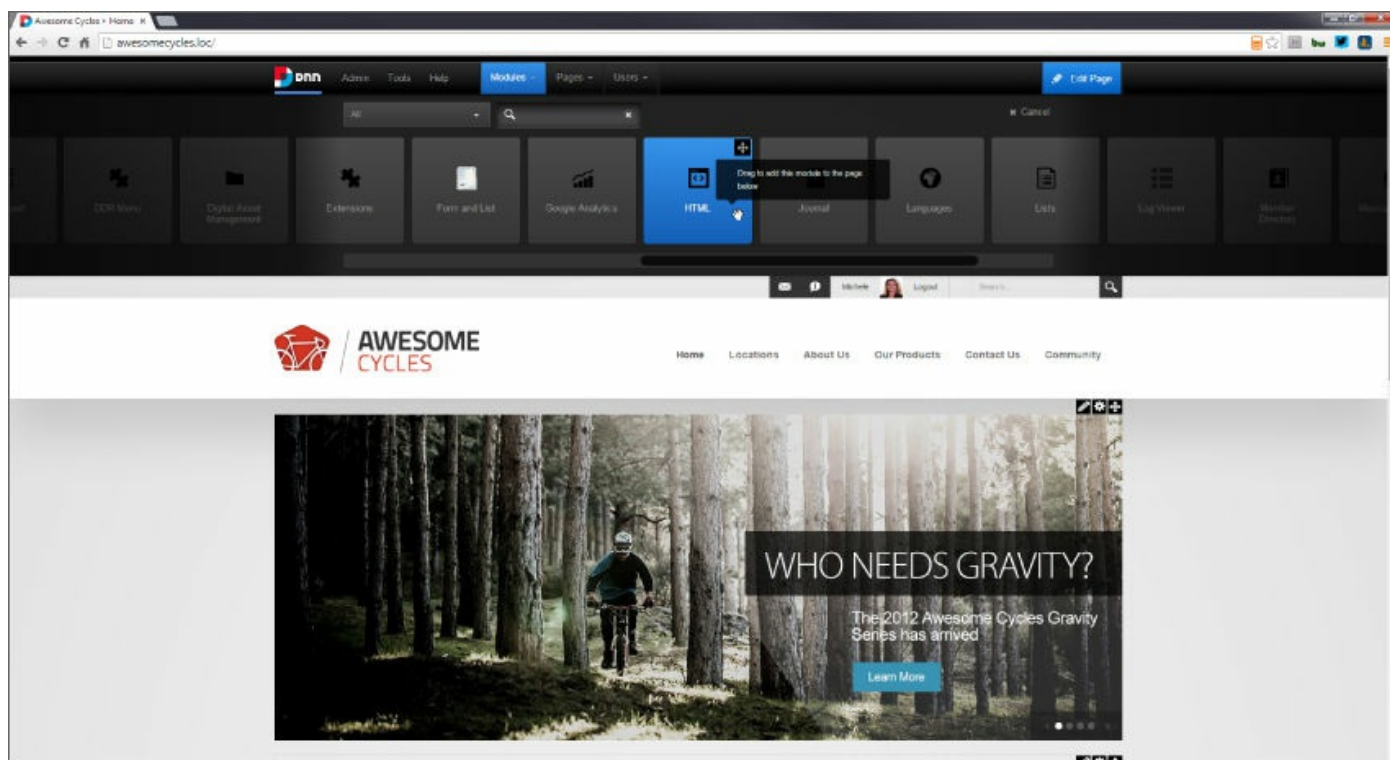


Figure 6.2

DNN version 7+ provides drag-and-drop functionality for adding modules to pages. After you select a module, simply click the module and start dragging it

onto the page below. As you drag the module down, the page's light blue regions come into view. These are the panes mentioned previously. The pane's name also comes into focus so that you know exactly on which pane you are focusing when hovering over it. While drag and drop is usually the easiest option, it is not the only option for getting a module onto the page. Users can also hover over the top-right corner of the modules in the drop-down menu where a small crosshairs icon is located. Hovering over this icon presents a different drop-down menu that lists all the panes that exist in the current page. To add the module to any specific pane, simply select the pane and click it.

One Module Across Multiple Pages

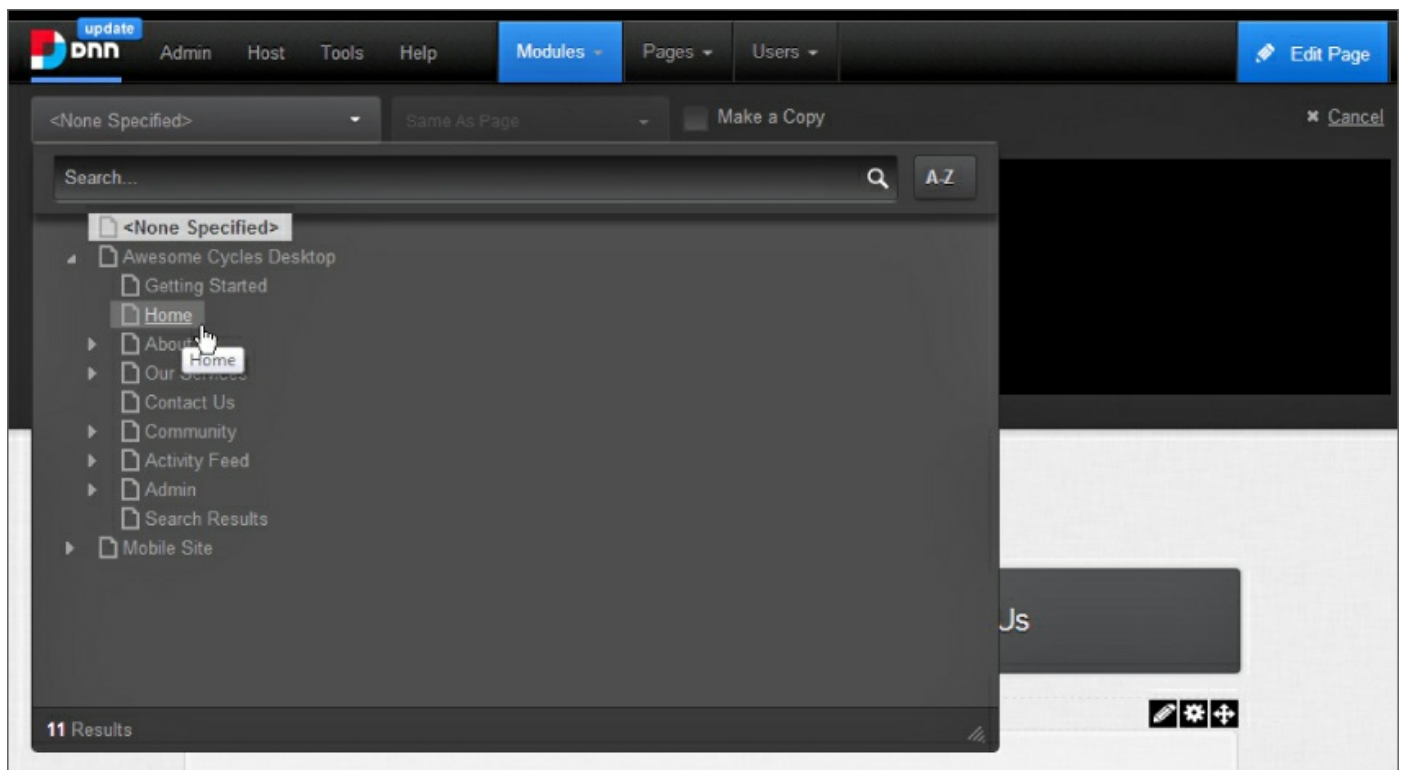
As noted earlier, modules are reusable pieces of functionality, and as such a single module can be inserted across multiple pages in a site. Say, for example, you have an HTML module that holds a banner image and you want that banner image to also exist on five specific pages in your site. Instead of adding a new module to all five of those pages and then inserting the same image, you can simply hover over the Modules menu item in the Control Panel and then click Add an Existing Module to the page. Adding already existing modules or sharing modules across multiple pages makes managing content that's repeated much more efficient. A content manager can then make edits in one module and instantly have those edits reflected across every page on which the module is shared.

When a user hovers over the Modules menu item just below the Add New Module option is the Add Existing Module option. This is the option you select when you want to add a module that already exists in the site to a different page. When opting to use Add Existing Module, you must first select the page to copy a module from, which will populate the drop-down list of modules. You can then follow the same process of adding a module to the page.

Also notable is that users can also elect to, at the point of adding an already existing module to the page, create a copy of the module so that the changes made in one module are not reflected in the original (and other shared instances of the module). When a user clicks Add Existing Module, a small checkbox at the top of the menu drop-down becomes visible that has the words Make a Copy beside it. Making a copy does exactly as it indicates and makes a copy of the module and in the process creates the module as a new instance of the module. By being a completely new instance, the module has its own settings and configurations, and its content is independent of the original version of the module.

One Module Across Multiple Sites

Just as you can share modules across multiple pages in a site, you can also share modules across multiple sites. DNN's Evoq Content solution has a feature called Site Groups, which allows host users to put multiple sites in a group. The Site Groups feature allows for single sign-on as well as a concept referred to as cross-site content sharing. Cross-site content sharing is essentially the act of sharing modules across totally different sites that exist in your instance of DNN. Once the host user creates a site group, you can go to Add Existing Module where you previously had to pick the page on which the module you wanted to add resided. You now have to start at the site level and pick the site in which the module you want to add resides. After picking the site, you follow the same process for picking the page and then adding the module to the page. In [Figure 6.3](#), you can see the interface that is presented when adding a module that exists across multiple sites.



[Figure 6.3](#)

Working with Modules

I've previously discussed what a module is, how modules can be added to pages, and how you can share them across pages and even sites. In this section, I focus on how you work with and configure modules once they're on the page.

In order to access a module's settings, you must be in Edit mode. To get into Edit mode, simply hover over the Edit Page menu item in the top-right corner of the Control Panel and click Edit this Page. This puts you into Edit mode where you will see small black icons in the top-right corner of every module on the page. These small black icons allow you to move the module on the page, access the module's settings, edit the module, delete the module, and access any specific functionality related to the module.

Once in Edit mode you will notice that DNN uses a very user-friendly and intuitive approach to managing content. This approach is called *in-context content management*, which means that whenever you want to edit content in DNN, you go to the location where the content resides and edit it right there in place. There's no “back office” or “content administration” area that you must go to in order to manage your content.

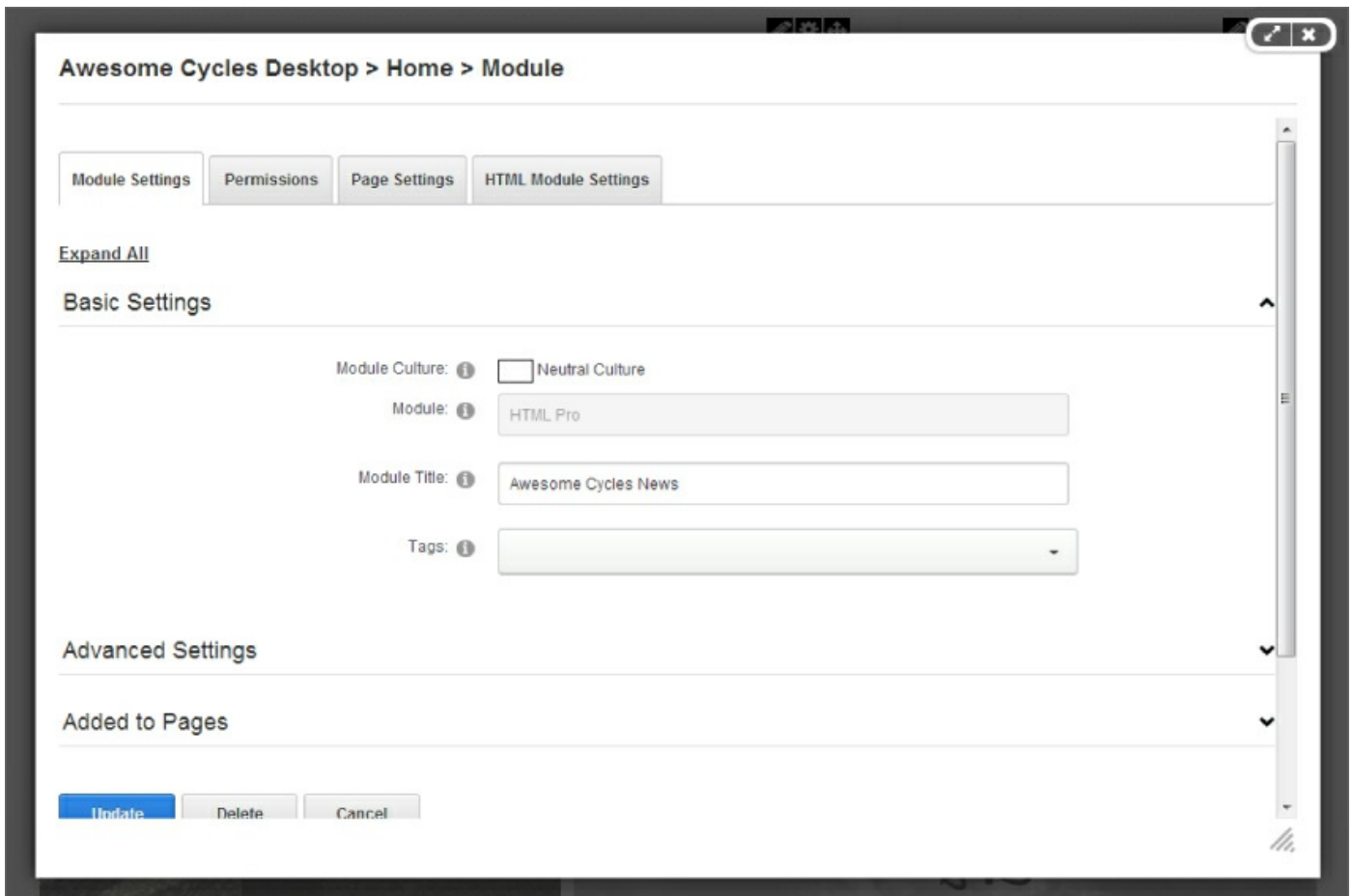
Moving Modules

Once a module is on the page, you can move its location again if needed. You can easily drag and drop the module to a new pane in a very similar fashion to how the modules are initially dragged and dropped on the page. You can also hover over the small black crosshairs icon in the top-right corner of the module action menu. When you hover over this menu, you see a list of panes that exist in the skin. To move the module, select the destination pane and click it. Your module is instantly moved to the new pane.

Module Settings and Action Menus

The small black icon that resembles a gear provides access to the module's settings menu. It also gives you access to import and export content (if the specific module supports importing and exporting), some help information, access to modify the module's code, the means to refresh the module, and the ability to delete the module. You will use several items in the settings menu, but you will probably access the module's settings more frequently than the others, so let's dive into the module settings a little more in depth.

When you click the Module Settings button, a pop-up window appears (unless the site's host user has disabled pop-ups). The pop-up window has three tabs that are always visible: Module Settings, Permissions, and Page Settings, as shown in [Figure 6.4](#). Most modules have a module-specific settings tab that appears as a fourth tab. Note that there can be more tabs if your site has been localized to handle multiple languages, at which point a Localization tab will appear.



[Figure 6.4](#)

The Module Settings Tab

We won't cover every specific detail of the module settings as several of the settings are straightforward, but I will discuss some of the more frequently used settings. In the basic settings area of the module's Settings tab, you can update the module's title, associate any tags you want with the module, and denote the module's culture in the case that your site has been localized.

In the Advanced Settings panel, you can select the Display on All Pages checkbox to have the module appear on every page in the site. If you check

this box, another checkbox will appear asking if you want to add the module on new pages only.

The Allow Indexing checkbox indicates that you want the contents of this module to be indexed by the search indexer so that the contents of the module appear in the site's search results. The Allow Indexing option is selected by default.

The next setting, Is Shareable, indicates whether you want to allow this module to be shared across multiple sites (as you previously walked through).

The View Only checkbox is visible only when the module's Is Shareable option is enabled. This box, when selected, enforces that the module is shareable only as a “view mode” of the module, meaning that the module cannot be edited on the other sites in which it is shared.

The Hide Admin Border option removes the admin border that's displayed around a module when the module is visible only to Admin users. When permissions are configured to where only Administrators can see the module, a light blue box with the message “Visible By Administrators Only” is displayed. This permission configuration is typical for the Administrative modules in the system. The Hide Admin Border option removes this message, which makes for better spacing on pages and eliminates the repeated messages about visibility.

The next two options, Header and Footer, provide an area where text or code can be placed just before or after the module's content. The last two options in the Advanced Settings area are the Start date and End date. You can set a date and even a time when you want the specific module's content to appear or to disappear. The scheduling of content is helpful whenever content is time sensitive in scenarios such as offering special content during the holidays, leading up to events, or for promotional time periods.

Just beneath Advanced Settings is the Added to Pages panel. This area shows the pages on which the module appears, which is important when a module is shared across multiple sites and/or pages. It gives you instant insight into the locations where the module is visible so that you know that any edits to this module will affect more than just the page currently being viewed.

The Permissions Tab

The Permissions tab displays the permissions grid. See [Figure 6.5](#). The

permissions grid gives administrators (and users with appropriate privileges) the ability to set permissions on the module. All modules have “Edit” and “View” permission, and some modules might provide additional items such as the Forms and Lists module, which adds granular editing options for its rows.

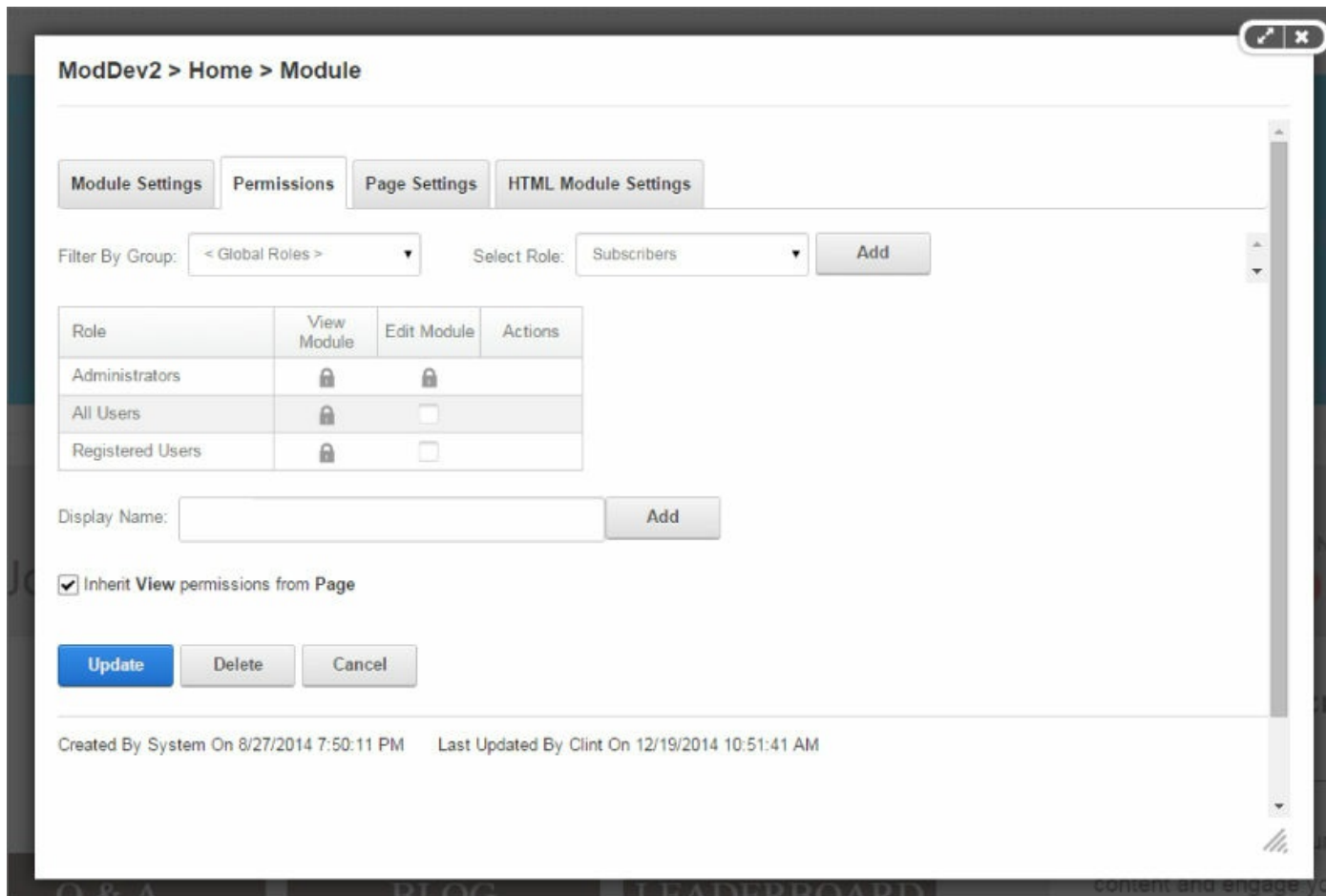


Figure 6.5

You may notice that some settings are locked by default (visualized by a lock icon), which includes the Admins row (admins always have all permission). The lock icons are present because the Inherit View Permissions from Page setting is enabled. By default, the view permissions for every module are set to inherit the view permissions from the page on which the module resides. This setting can be easily disabled if desired, and when disabled, the lock items are removed.

In order to add other security roles into the grid, click the drop-down menu in the Select Role area, select a security role, and click the Add button. This adds the security role into the grid, and you can configure permissions for that security role. If desired, individual users can also be added into the permissions grid by typing a user's display name in the Display Name input

box just below the permissions grid. Clicking Add will inject the individual user into the grid, and then you can configure permissions for that individual user.

Remember that in DNN the permissions are set both at the page level as well as at the module level. Having the combination of permissions set at the page and module levels provides very granular permissions capabilities, as admin users can allow specific users or groups of users (security roles) to see or not see, and edit or not edit, the entire page or even specific modules on the page. This is helpful in scenarios where you need specific users to have rights to edit certain pages or even modules on the pages. Content editors then log in to the site and have access to edit only the regions of the site where they have been granted edit permissions.

DNN provides additional permissions, which are not exposed by default in DNN Platform. To display them, you will need to install a Granular Permission Provider, like the one included with Evoq Content.

The Page Settings Tab

The Page Settings tab provides several other settings for the module, which are specific just for the current page. When modules are shared across multiple pages, these settings allow for overriding the settings from the original module. We won't cover every specific setting in this section in detail as some settings are straightforward. Here are the options of the Page Settings tab of the module. Also, note that while this list is present and visible in the user interface, some of these options have been deprecated. I've noted the items that are deprecated.

- **Icon:** Controls the icon displayed for containers that implement the icon token.
- **Alignment:** Controls the alignment of a module's contents. (deprecated)
- **Color:** Controls the background color to be applied to a module's contents. (deprecated)
- **Border:** Controls the width of the border around the module's contents. (deprecated)
- **Collapse/Expand:** Controls the default visibility of the module—whether it is collapsed or expanded.
- **Display Container:** Allows you to indicate whether the module's

container is shown.

- **Allow Print:** Controls the display of the print icon for each module. (deprecated)
- **Allow Syndicate:** Controls the setting to allow RSS feeds to be generated for modules that implement the ISearchable interface. (deprecated)
- **Is a WebSlice:** Allows you to make the module an IE8 webslice. (deprecated)
- **Module Container:** Allows users with edit permissions to choose the container for the module. If left to the default, the module will display the parent page's or site default container.

You can use the Display Container setting to indicate whether you want the module's container to be shown. Modules are wrapped in something called a container, which is usually part of a skin/theme package. Containers might do things like add a title, border, background, or specific text font to the module. Using the Display Container setting, you can hide the container's styles and effects. Generally speaking, it is preferred to change the container to a different container rather than hide the module's container.

The last option in the Basic Settings area is the Module Container drop-down menu. The default container is set at a site-wide, global level, but you can override this at the page level and at the individual module level, which is the very reason this setting is available. An example scenario where this may be useful is that when you need to display a warning, you could override the module container at the module level and make use of a container that displays a red frame around the module.

Just below the Basic Settings panel is the Cache Settings panel where you can set the module's cache provider. Available options for this are File and Memory. This chapter is not designed to explain cache in depth, but at a high level we can all agree we want our websites to load quickly, and this is where caching becomes beneficial. Caching allows us to store objects so that we can reduce the number of calls to the database, thus making our pages load more quickly. Content that doesn't change frequently such as a list of countries or states is a great candidate for being cached. Choosing the File option stores cached objects in a physical file on the server, whereas choosing the Memory option holds the cache in the server memory. Generally speaking, if your DNN site is hosted in a web farm you would opt for the “file” option and if

your DNN site is hosted on a single server, then you would opt for “memory.” After you select your caching provider, you can then set the Cache Duration time in the Cache Duration text box just beneath the caching provider.

Where to Get Modules

Now that you're familiar with what a module is, it's helpful to know where modules can be found online and how you can also custom develop your own modules. I'll first discuss some online locations where you can access modules.

The DNN Store

The most common place that people look for modules is on the DNN Store (<http://Store.DNNSoftware.com>). The store is an online marketplace where module and skin/theme vendors sell their modules and skins/themes. There are thousands of modules and skins available for you in the store. The selection of modules in the store is very robust, and also notable is that not only do many module vendors allow you to buy the install package for the module, but some also let you purchase the source code for their modules. Not all vendors have this option, but a good number of them do. This allows you to further extend and customize a module's functionality to meet your specific requirements. Also, by having the source code for the module, you don't have to worry if something happens and the vendor vanishes. By having the source code, you are able to further develop the code if needed in the future. The store also has “review” functionality that allows people who purchased modules to review and rate the modules. When looking to purchase a module, it's usually helpful to look at the rating and reviews.

Forge

While module vendors run businesses and make their livings by selling modules, some module developers will give you their modules for free. You can even find some free modules in the store as well. Often developers create modules for personal use, and they don't mind allowing others to use the module, but they just don't want to charge money for it or provide overly extensive support for the module. This type of occurrence is common for an open source project like DNN. People want to contribute back, and donating modules is one way this giving back occurs. Modules are not the only extension type found in the Forge. You can also download skins/themes and providers from the Forge. Also, all of what were previously known as the “core modules” are now listed on the Forge. You can find the Forge at www.dnnsoftware.com/forge. So if you're looking for some functionality, you


should definitely check out the Forge, as you may be able to find a helpful module there. A host user can browse both the store and the Forge from within the DNN site. A host user who travels to [Host](#) [Extensions](#) [More Extensions](#) can search for and instantly deploy modules from the store or Forge.

Developing Your Own Modules

Developers love DNN because they can quickly get to the heart of their custom functionality and business logic by leveraging DNN's core capabilities. Developers need not worry about coding out authentication, permissions, security, and so on, because all that is handled by DNN. This reduces the time it takes to deploy custom applications because developers can focus on their specific module's requirements and functionality while inheriting the rest of the functionality from DNN. Once developed, a module can be easily installed into the site via DNN's extension installer.

The only requirements for developing your own module are the knowledge of ASP.NET and the DNN API. Modules are usually built inside of Visual Studio, and templates that can speed up this process for developing a module are also available. We're not going to delve into the details of module development as that is covered [Chapters 13](#) through [16](#). Ultimately, the thing to know is that with knowledge of ASP.NET and DNN and with the Visual Studio software, a developer can create a custom module.

Viewing Modules and Extensions

Selecting Host  Extensions opens the Extensions page. The Extensions page has the Installed Extensions tab where lists of all extensions are visible in the system. There are various extension types visible such as authentication systems, providers, language packs, skins, widgets, and modules. These various extension types are organized in panels, and the Modules panel is expanded by default. When you click a panel, it expands to show all installed extensions of that specific type. All modules shown on this tab are already installed into your site.


The next tab, Available Extensions, shows extensions that are available to be installed but are not yet currently deployed in your site. In this tab, various extension types are presented, again organized by type.

The third tab, Purchased Extensions, displays a list of modules that you have purchased from the store. You also can download and install to your site from this view.

Finally, the More Extensions tab shows the interface where you can search for modules from both the Store and the Forge directly from within your site.

Installing Modules into DNN

Regardless of whether you custom developed your module, bought it from the DNN Store, or downloaded it from the DNN Forge, eventually you have to install the module into your site. Since DNN is built with modular functionality in mind, you can install the module into any DNN site and use it. This is a benefit of the modular architecture of DNN and the reusable nature of modules; you can develop a module once and install it into any number of DNN sites.

Typically, a module install package comes in the form of a Zip file. In order to install this into your site, you need to be logged in as the host user. After authenticating as the host user, you should access Host  Extensions, which was discussed in the previous section. On that page you'll see the Install Extension Wizard menu item just above the tabs of the page. This button is generally highlighted in blue. Just click Install Extension Wizard and walk through the process of installing the module. This process consists of a few screens where you must accept license terms as well as the final screen where you can see the results of the module's install into your site. If all goes well, a Successful message appears at the bottom of the install script and then a button that says Return. Clicking Return recycles the site's application pool because at this point DNN recognizes that the just installed module placed a new .dll file in the site's bin folder. (The application pool is a group of one or more URLs that are served by a worker process. Your DNN site has its own app pool.) DNN automatically recycles the app pool for the site. For this reason the next page load after clicking Return can sometimes be a little slower than simply navigating around the pages of the site.

The Extension Verification System

The Extension Verification System (EVS) (<http://EVS.DnnSoftware.com>) is a relatively new service from DNN that helps extension developers and any users who are thinking of installing an extension into their site test the extension before installing. The EVS tests any type of extension, but the focus here is on the module extension type in this chapter.

Upon uploading a module into EVS, the service puts the module through an extensive set of tests to determine any errors or issues the module may have. Once the system completes the verification process, a view with all results is shown. The EVS results denote several properties of the module such as the minimum version of DNN the module can run on, whether the module is Azure compatible, and whether the module has errors present. EVS also has panels that can be expanded to show even more information about any errors or warnings that EVS may have found while processing the module.

EVS is a great resource for those who may be thinking of buying a module and installing into their site because you can run the module through EVS before installing into your site and denote if the module is capable of running on your site's version of DNN. Likewise for developers, EVS can be beneficial because it may find some issues in a module of which the developer may not have been aware. To help communicate the benefit of the EVS service we have described the EVS printout as being similar to a Carfax that is often requested before buying a car.

In Depth with the HTML Module

There are thousands of modules that we could review, but in this section we review the most commonly used module across the majority of DNN websites: the HTML module. Typically, there will be some HTML content displaying images and/or text on nearly every page, and this is where the HTML module comes into play. It should be easy to see why it's important to have a very flexible, powerful, and robust feature set in the HTML module.

By default, any time you create a new page in DNN there is automatically one HTML module placed on the page. As soon as you go into Edit mode on the page, you will see the HTML module on the page. Typically, you'll see the black module action menu icons as well as the text Enter Title on the page indicating that there is an HTML module awaiting configuration and content.

Hovering over the left-most module action menu icon (the pencil icon) option, Edit Content becomes visible. Click Edit Content to open a pop-up window, and a rich-text editor is loaded into the pop-up. The editor gives an accurate representation of what the content will look like once placed on the page and thus makes it easy for content managers to manage content. [Figure 6.6](#) illustrates the text editor of the HTML module.

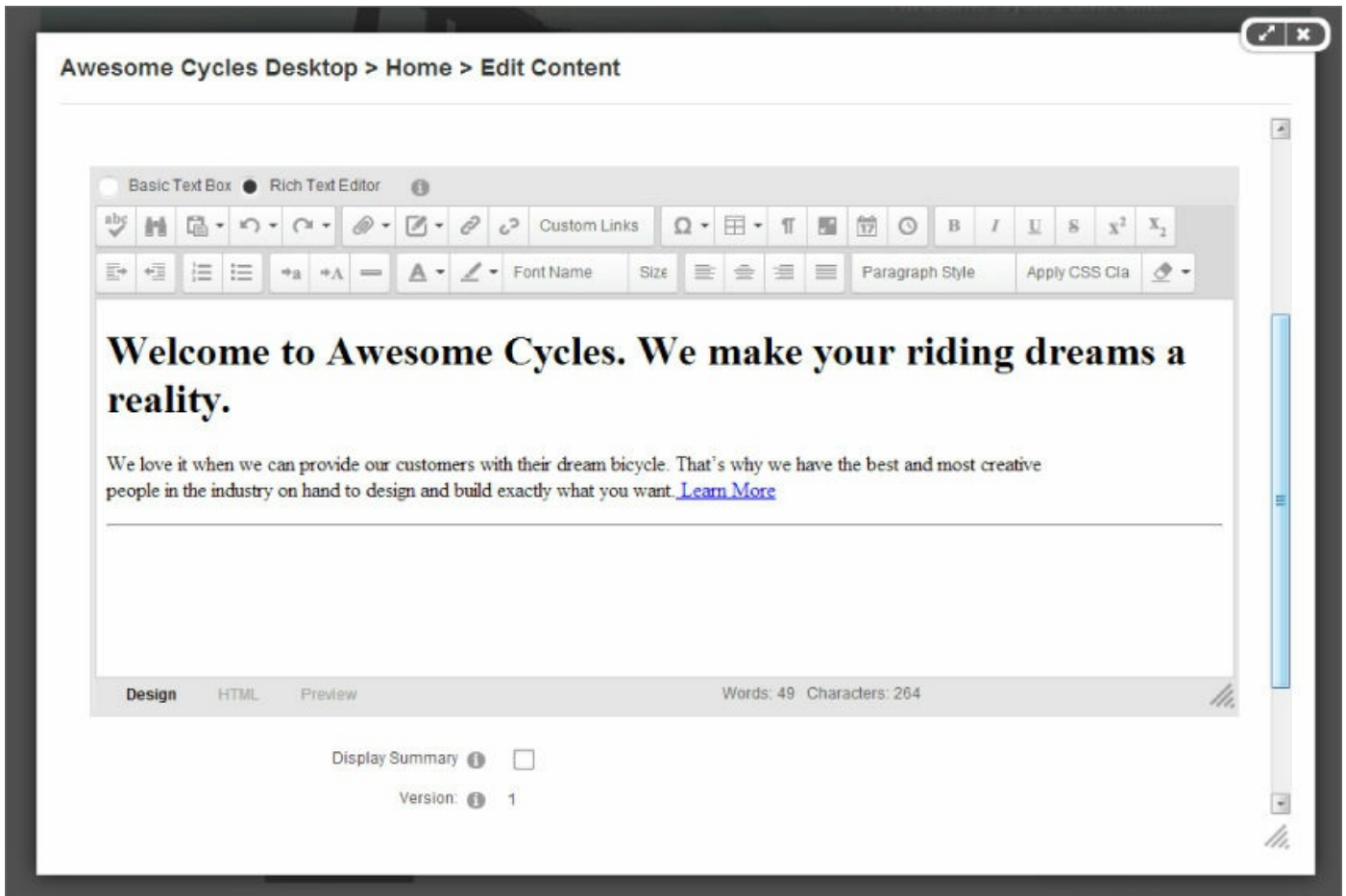


Figure 6.6

The HTML editor allows content managers to easily add and update content through inserting text, inserting images, creating tables, creating hyperlinks, using templates, spell-check, find and replace, and much more. With appropriate permissions configured, content managers can easily upload new images and documents right from within the editor. There is also an HTML view (sometimes referred to as “code” view) that content managers can toggle into if they want to see the code created by their content. Some content managers feel more comfortable with code and use code view to get in and update their HTML code from the code view perspective.

Administering the HTML Module

Some modules have so many options and configuration possibilities that they merit their own administrative area. The HTML module is one of these types of highly configurable modules. Only the host user has the permissions to create a new configuration for the HTML module.

As previously shown, the HTML editor has several options available to

content managers. However, sometimes site managers have specific security roles or specific pages where they don't want users to have all editing capabilities. A common usage of this is to restrict some users from being able to upload documents and images while others have the ability to upload these types of files. Through the HTML Editor Manager option of the Host menu's Advanced menu, host users can remove any specific functionality that they want from the HTML editor.

By default, there is an editor configuration for Everyone that exists in your DNN site. To create a new configuration, simply click the Everyone configuration and at the bottom click Copy. Clicking Copy starts the process of creating a new HTML configuration. The first step you must take is to select the configuration's binding. The host user can bind this configuration to a specific page, to a specific security role, or across the whole site.

Once the new HTML editor configuration is created, the host user can configure several settings within the newly created configuration. There are too many settings to explain here, but a few of these settings are editor's skin, edit modes (Design, HTML, Preview), the option to automatically remove scripts, usage of specific styles within the editor, the maximum file size for uploaded images and documents, as well as restricting specific file extensions that can be uploaded. Beyond the settings, the host user can also remove any HTML editor functionality from the editor's user interface. Typically a content editor sees two rows of icons representing the edit functionality within the HTML editor. However, the host user can remove any of these items and show the specific icons (functionality) only to the desired users.

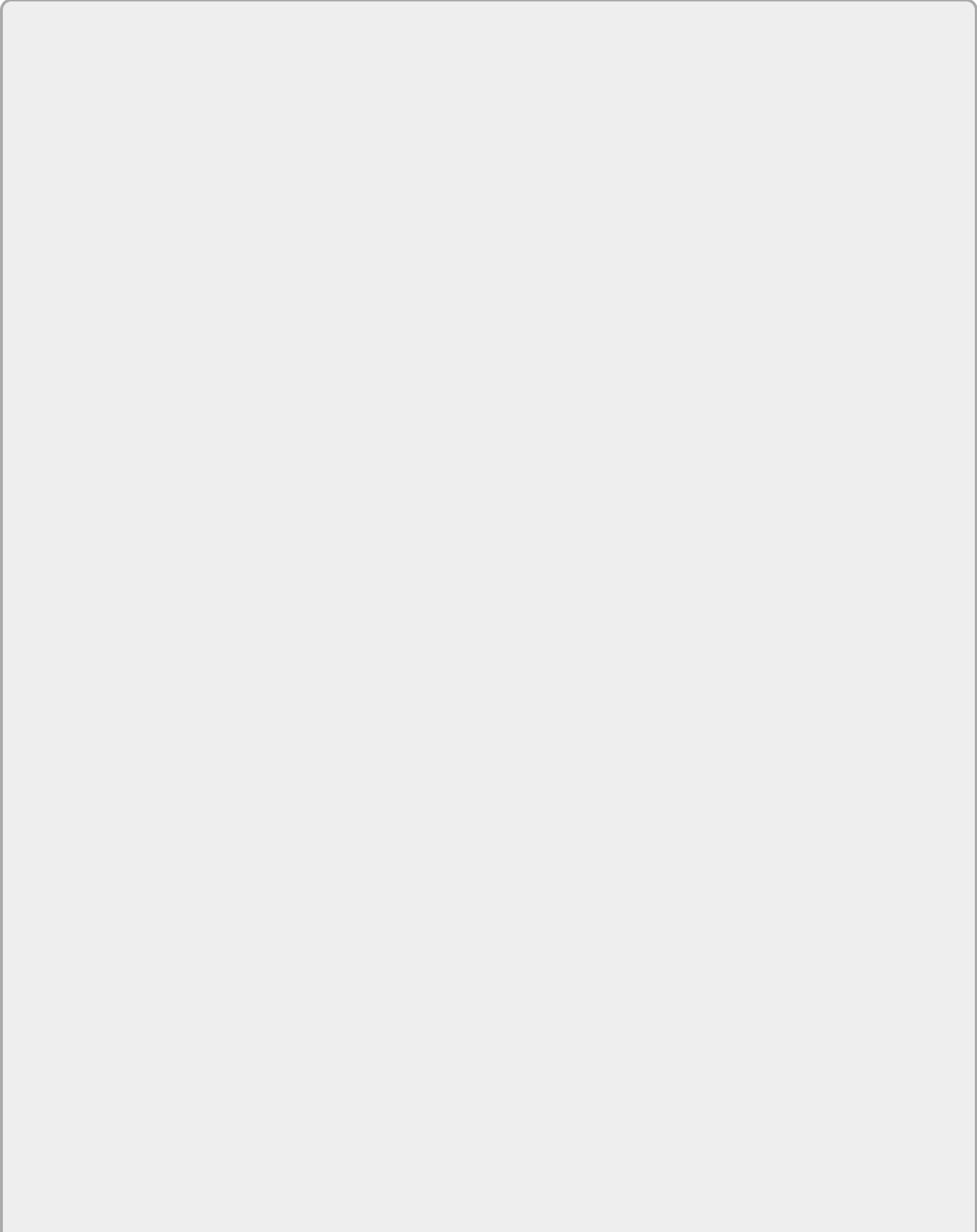
HTML Pro Module Features in Evoq Content

The aforementioned capabilities exist within the HTML module in the DNN Platform out of the box. Additionally there are advanced features of the HTML module that are available only in the Evoq Content solution. The features that are unique to the Evoq Content solution are mentioned more in-depth in [Chapter 20](#).

Summary

This chapter has explored modules, the most popular extension point in DNN, at a high level. You should now understand what a module is, where modules can be purchased and from where they can be downloaded, how to install them in your site, how to add them to pages in your site, and how to work with modules on your pages. You've also seen the most commonly used module in DNN, the HTML module. You should now feel confident in your ability to work with modules and be able to manage basic content on your site.

Chapter 7
System Architecture



What's in this chapter?

- Delving into patterns
- Examining the DNN architecture
- Exploring the basic layers in DNN

The initial version of DNN (or DotNetNuke) was derived from the IBuySpy Portal starter kit that was released in both a C# version and a VB.NET version. IBuySpy was designed to provide a real-world example of some of the new development concepts of the ASP.NET platform.

While DotNetNuke 7 still retains some of the core principles that were introduced in IBuySpy Portal, if you look at the source of DNN today, it doesn't really resemble IBuySpy. Actually, DNN 7 bears little resemblance to the earliest versions of DNN. Since DNN 6, the application is written in C#, while earlier versions were written in VB.NET.

The key technologies used in DNN have changed over the years. DNN 7 uses the following technologies in its supported architecture:

- Microsoft Internet Information Services (IIS) version 7 or later running in Integrated Pipeline mode.
- An operating system that supports IIS 7 (that is, Windows Vista, Windows 7 or Windows 8, Windows 2008 Server, or Windows Server 2012).
- The ASP.NET framework. (Early versions of DNN ran on the then-current version of ASP.NET. DotNetNuke version 7 requires ASP.NET version 4.0 or later for both runtime and development.)
- C# (Visual Basic.NET or any other CLR-compliant language can be used to write extensions to the platform.)
- ASP.NET Web forms.
- ASP.NET Web API.
- WebFormsMVP framework.
- PetaPoco micro-ORM.
- JQuery, JQuery-UI, and Knockout JavaScript frameworks.

- Microsoft SQL Server 2008 (Express, Standard or Enterprise) or Microsoft SQL Server 2012 (Express, Standard or Enterprise).

In addition, DotNetNuke uses a number of key design patterns and concepts:

- Provider Pattern
- Service Locator Pattern
- Inversion of Control Container
- Repository Pattern using Entities and Controllers
- Centralized Custom Business Object Hydration
- Model View Presenter and Module View Controller Patterns

Patterns and Concepts

DNN uses a number of key concepts and patterns. Patterns are architectural recipes that help developers to write certain types of functionality in a reliable fashion. For example the Provider Pattern is used to provide a way to extend the DNN Platform. The Service Locator Pattern and Inversion of Control Container are used to provide dependency inversion, which allows the platform to be built in a modular fashion and supports testability.

Provider Pattern

The Provider Pattern is a design pattern that was formalized in ASP.NET 2.0 to provide extension points for an application. DNN has adopted this pattern throughout the framework. In the Provider Pattern, the API (Application Programming Interface) itself is separated from the implementation of the API. The basic functionality is defined in an interface or more commonly an abstract base class. The actual implementation is then defined in a concrete implementation.

This fundamental design concept is not new. The Provider Pattern allows for a default implementation, but it also allows developers to create alternate implementations.

Several areas in DNN use the Provider Pattern:

- Data Provider
- Membership Provider
- Profile Provider
- Roles Provider
- Navigation Provider
- Caching Provider
- Scheduling Provider
- Logging Provider
- HTML Editor Provider
- Search Provider
- Friendly URL Provider

- Folder Provider

Most providers use configuration to determine the default implementation. This configuration information is usually placed in the web.config file, although some providers use more advanced configuration.

An example of the Folder Provider configuration section follows:

```
<folder defaultProvider="StandardFolderProvider">
  <providers>
    <clear/>
    <add name="StandardFolderProvider"
        type="DotNetNuke.Services.FileSystem.StandardFolderProvider,
DotNetNuke"/>
    <add name="SecureFolderProvider"
        type="DotNetNuke.Services.FileSystem.SecureFolderProvider,
DotNetNuke"/>
    <add name="DatabaseFolderProvider"
        type="DotNetNuke.Services.FileSystem.DatabaseFolderProvider,
DotNetNuke"/>
  </providers>
</folder>
```

The Provider API requires that providers define a name and type. In addition, a default provider must be defined. The Folder Provider is quite simple, but some providers require additional settings. An example of a provider that requires such additional settings is the Data Provider.

```
<data defaultProvider="SqlDataProvider">
  <providers>
    <clear/>
    <add name="SqlDataProvider"
        type="DotNetNuke.Data.SqlDataProvider, DotNetNuke"
        connectionStringName="SiteSqlServer"
        upgradeConnectionString=""
        providerPath="~\Providers\DataProviders\SqlDataProvider\"
        objectQualifier="dnn_"
        databaseOwner="dbo"/>
  </providers>
</data>
```

Providers are called using an `Instance` method on the base type:

```
var defaultProvider = DataProvider.Instance();
```

This `Instance` method returns the default provider. If a specific instance is required, then the `Instance` method has an overload that takes the name of the instance:

```
var secureProvider = FolderProvider.Instance("SecureFolderProvider");
```

Prior to DNN 5, the providers were loaded when first referenced using a Singleton pattern, but since DNN 5, the providers have been loaded on Application Start and loaded into an Inversion of Control (IoC) container. The `Instance` methods then retrieve the appropriate instance from the container.

Service Locator Pattern

The Service Locator Pattern is similar to the Provider Pattern. In the Service Locator Pattern, an interface is used to define the API. As an example, [Listing 7.1](#) shows the `IJavaScriptLibraryController` interface that defines the API used to access `JavaScriptLibrary` entities.

[Listing 7.1](#): The `IJavaScriptLibraryController` Interface

```
public interface IJavaScriptLibraryController
{
    void DeleteLibrary(JavaScriptLibrary library);

    JavaScriptLibrary GetLibrary(Func<JavaScriptLibrary, bool>
predicate);

    IEnumerable<JavaScriptLibrary>
GetLibraries(Func<JavaScriptLibrary,
                bool> predicate);

    IEnumerable<JavaScriptLibrary> GetLibraries();

    void SaveLibrary(JavaScriptLibrary library);
}
```

Unlike with the Provider Pattern, there really isn't any intention to support multiple implementations of the interface, except that the implementation as used in DNN 7 supports the concept of a testable instance so that the developer can write code that is testable.

As with the Provider Pattern, the Service Locator Pattern provides an `Instance` method (actually the `ServiceLocator` base class implements this as a property, but the concept is the same) so developers can make calls like

```
var libraries = JavaScriptLibraryController.Instance.GetLibraries();
```

[Listing 7.2](#) shows the `JavaScriptLibraryController` class. The Service Locator

Pattern also provides a `GetFactory` factory method that determines the instance returned. Under the covers the `ServiceLocator` base class manages a Singleton instance, and the `GetFactory` method is called if there is no instance to return.

There is no configuration required for the Service Locator Pattern and so it is used in situations where extensibility is not required.

Listing 7.2: The JavaScriptLibraryController Class

```
public class JavaScriptLibraryController
    : ServiceLocator<IJavaScriptLibraryController,
JavaScriptLibraryController>
    , IJavaScriptLibraryController
{
    protected override Func<IJavaScriptLibraryController>
GetFactory()
    {
        return () => new JavaScriptLibraryController();
    }
}
```

In order to support testing, the `ServiceLocator` base class also provides a `SetTestableInstance` method. This allows Unit Test writers to create a mock of the interface and use it in the unit test.

```
var mockSearchHelper = new Mock<ISearchHelper>();
SearchHelper.SetTestableInstance(_mockSearchHelper.Object);
```

Inversion of Control (IoC) Container

DNN includes a simple Inversion of Control Container. In simplest terms, an Inversion of Control Container is an in-memory store of objects that can be accessed at any time. As the Provider Pattern supports multiple implementations, the different implementations are loaded into the container on application start.

Developers don't need to worry about explicit dependencies. They can program against an interface or abstract base class, and the Inversion of Control Container takes care of determining which implementation to return.

Let's look at how the Provider Pattern uses the IoC Container to return the correct provider instance. During application start the container is created:

```
ComponentFactory.Container = new SimpleContainer();
```

Next, the providers are instantiated based on the web.config settings:

```
ComponentFactory.InstallComponents(new ProviderInstaller("data",  
typeof(DataProvider),  
typeof(SqlDataProvider)));
```

Under the covers the `InstallComponents` method registers all the providers that implement the Data Provider API. Finally, the `DataProvider` class's `Instance` method retrieves the default `DataProvider` instance by calling the `GetComponent` method of the `ComponentFactory` class:

```
public static DataProvider Instance()  
{  
    return ComponentFactory.GetComponent<DataProvider>();  
}
```

Repository Pattern

When dealing with the objects of the business layer, DNN uses a variation of the Repository Pattern. DNN's objects are lightweight—they are a blueprint or representation of an entity that is important to the application. These entities have properties, but most of the time they have no methods. The business logic in DNN is encapsulated in additional classes. These are often called repositories, but DNN uses the name “Controller” to refer to the business logic classes. This naming is historical, but the Controller classes are essentially Repository classes.

You have already seen an example of a DNN Controller class (`JavaScriptLibraryController` and its companion interface `IJavaScriptLibraryController`), which contains the methods that manipulate `JavaScriptLibrary` entities.

[Listing 7.3](#) shows the `JavaScriptLibrary` entity class.

Listing 7.3: The JavaScriptLibrary Class

```
[Serializable]  
public class JavaScriptLibrary : IXmlSerializable  
{  
    public int JavaScriptLibraryID { get; set; }  
    public int PackageID { get; set; }  
}
```

```

public string LibraryName { get; set; }
public Version Version { get; set; }
public string ObjectName { get; set; }
public string FileName { get; set; }
public ScriptLocation PreferredScriptLocation { get; set; }
public string CDNPath { get; set; }
#region IXmlSerializable Implementation
public XmlSchema GetSchema()
{
    throw new NotImplementedException();
}
public void ReadXml(XmlReader reader)
{
    while (reader.Read())
    {
        if (reader.NodeType == XmlNodeType.EndElement)
        {
            break;
        }
        if (reader.NodeType == XmlNodeType.Whitespace)
        {
            continue;
        }
        else
        {
            switch (reader.Name)
            {
                case "javaScriptLibrary":
                    break;
                case "libraryName":
                    LibraryName =
reader.ReadElementContentAsString();
                    break;
                case "objectName":
                    ObjectName =
reader.ReadElementContentAsString();
                    break;
                case "fileName":
                    FileName =
reader.ReadElementContentAsString();
                    break;
                case "preferredScriptLocation":
                    var location =
reader.ReadElementContentAsString();
                    switch (location)
                    {
                        case "BodyTop":
                            PreferredScriptLocation =
ScriptLocation.BodyTop;
                            break;
                        case "BodyBottom":

```

```

        PreferredScriptLocation =
ScriptLocation.BodyBottom;
        break;
        default:
        PreferredScriptLocation =
ScriptLocation.PageHead;
        break;
    }
    break;
    case "CDNPath":
        CDNPath =
reader.ReadElementContentAsString();
        break;
        default:
            var content =
reader.ReadElementContentAsString();
            break;
    }
}
}
}
public void WriteXml(XmlWriter writer)
{
    writer.WriteStartElement("javaScriptLibrary");
    writer.WriteElementString("libraryName", LibraryName);
    writer.WriteElementString("fileName", FileName);
    writer.WriteElementString("objectName", ObjectName);
    writer.WriteElementString("preferredScriptLocation",
        PreferredScriptLocation.ToString());
    writer.WriteElementString("CDNPath", CDNPath);
    writer.WriteEndElement();
}
#endregion
}

```

While it is not true that the `JavaScriptLibrary` class has no methods, the only methods it contains are methods that implement the `IXmlSerializable` interface.

Architectural Overview

By using a multitiered architecture, the application can be distributed across the web server and the database server, as shown in [Figure 7.1](#).

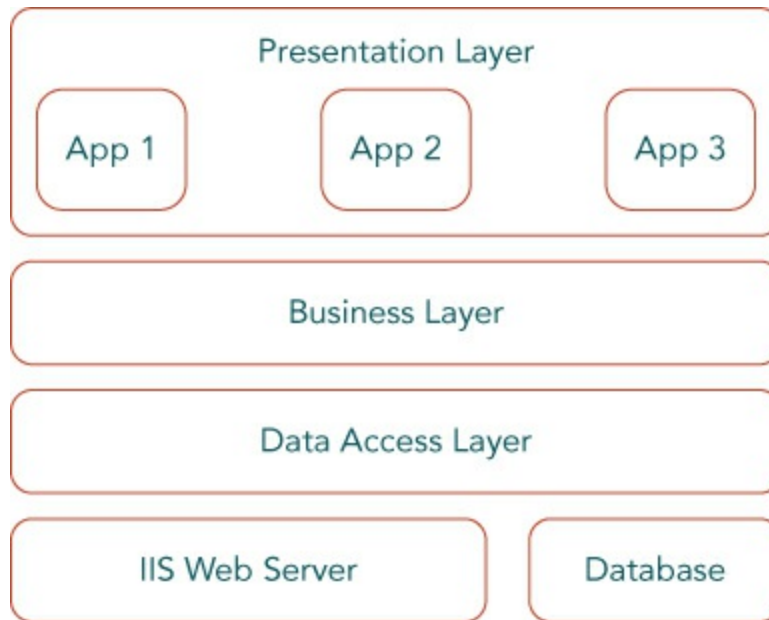


Figure 7.1

The web server contains the Presentation, Business, and Data Access Layers, and part of the Data Layer. The database server contains the rest of the Data Layer.

Data Layer

The Data Layer sits at the bottom of the multilayer framework. As DNN uses the Provider Model extensively (see earlier discussion), and in particular it includes an abstract Data Provider, in theory the Data Layer could be implemented for many different databases. In practice, while there have been MySQL and Oracle implementations, the only Data Provider that has been supported continuously is a Microsoft SQL Server implementation of the Data Provider API.

DNN includes a number of script files that are executed during installation or upgrade. These scripts can be found in the folder

`Providers/DataProviders/SqlDataProvider`. The actual scripts executed depend on the version of DotNetNuke being installed. These scripts create the database tables, stored procedures, and data necessary to run DNN. There are two classes of scripts: install scripts and upgrade scripts.

Install Scripts

There are two installation scripts that are run only during a new installation. These scripts are as follows:

- **DotNetNuke.Schema.SqlDataProvider:** Installs the tables, views, and stored procedures
- **DotNetNuke.Data.SqlDataProvider:** Fills the tables with any default data

These scripts are updated only at major versions (for example, DNN 5.0, DNN 6.0, and DNN 7.0).

Upgrade Scripts

The second group of scripts are upgrade scripts. These scripts make any changes to the schema or base data that are required to upgrade from one version to the next. The file-naming convention includes the version of the script expressed in the format `ox.oy.oz` followed by the `SqlDataProvider` extension.

For example, the script to upgrade from 7.0.0 to 7.0.1 is named `07.00.01.SqlDataProvider`.

As the install scripts are updated only at major versions, these upgrade scripts may be executed during either install or upgrade.

For example, installing DNN version 7.0.3 executes the following scripts:

- `DotNetNuke.Schema.SqlDataProvider`
- `DotNetNuke.Data.SqlDataProvider`
- `07.00.01.SqlDataProvider`
- `07.00.02.SqlDataProvider`
- `07.00.03.SqlDataProvider`

Upgrading from version DNN 6.2.8 (the last version prior to the release of DNN 7) to the same version executes the following scripts:

- `07.00.00.SqlDataProvider`
- `07.00.01.SqlDataProvider`
- `07.00.02.SqlDataProvider`

- 07.00.03.SqlDataProvider

Script Syntax

SQL Data Provider scripts are written in T-SQL, but there are two important non-SQL tokens used in them:

- {databaseOwner} defines the database owner to append to data objects in the scripts.
- {objectQualifier} defines a string to prefix the data objects within the scripts.

Both of these tokens represent a programmatically replaceable element of the script. They are defined in the data element of the web.config file, which you saw previously but is reproduced here:

```
<data defaultProvider="SqlDataProvider">
  <providers>
    <clear/>
    <add name="SqlDataProvider"
        type="DotNetNuke.Data.SqlDataProvider, DotNetNuke"
        connectionStringName="SiteSqlServer"
        upgradeConnectionString=""
        providerPath="~\Providers\DataProviders\SqlDataProvider\"
        objectQualifier="dnn_"
        databaseOwner="dbo"/>
  </providers>
</data>
```

For example, the T-SQL code in [Listing 7.4](#) from the 07.00.06.SqlDataProvider script shows how the `GetDatabaseInstallVersion` stored procedure is created.

[Listing 7.4](#): The Script to Create the `GetDatabaseInstallVersion` Stored Procedure

```
IF EXISTS (SELECT * FROM dbo.sysobjects WHERE id =
object_id(N'{databaseOwner}
[{objectQualifier}GetDatabaseInstallVersion]')
AND OBJECTPROPERTY(id, N'IsPROCEDURE') = 1)
    DROP PROCEDURE {databaseOwner}
[{objectQualifier}GetDatabaseInstallVersion]
GO
CREATE PROCEDURE {databaseOwner}
```

```

[{{objectQualifier}}GetDatabaseInstallVersion]
AS
    SELECT TOP 1
        Major,
        Minor ,
        Build
    FROM      {databaseOwner}{objectQualifier}Version V
    WHERE     VersionId IN
        (SELECT MAX(VersionId) AS VersionID
         FROM   {databaseOwner}[{{objectQualifier}}Version]
         GROUP BY CONVERT(NVARCHAR(8), CreatedDate,
112)
        )
GO

```

This code looks like T-SQL with the addition of the two non-SQL tags. The first block of T-SQL checks if the stored procedure exists, and if it does, the stored procedure is dropped. The check for existence is important: DNN developers try to ensure that the scripts can be reexecuted, so checking for the existence of an object before deletion makes the code more resilient.

The next line creates the new stored procedure:

```

CREATE PROCEDURE {databaseOwner}
[{{objectQualifier}}GetDatabaseInstallVersion]

```

The new stored procedure is created in the context of the `databaseOwner` as defined in `web.config`, and the name of the stored procedure is prefixed by the `objectQualifier` value from `web.config`.

If you replace the tokens with the values from the example `web.config` code (seen previously), then the previous code would be converted to

```

CREATE PROCEDURE dbo[dnn_GetDatabaseInstallVersion]

```

The `objectQualifier` attribute is useful when you have only one database but want to support more than one application. As many applications have the same names for database objects, this allows you to separate the DotNetNuke database objects from the other application's objects.

Data Access Layer

The Data Access Layer provides the interface between the Data Layer and the Business Layer. It allows for data to flow to and from the data store.

As mentioned previously, the Data Access Layer uses the Provider Pattern.

The Data Access Layer consists of two classes:

- **DataProvider**: An abstract base class that establishes the contract that the implementation of the API must fulfill
- **SqlDataProvider**: A concrete class that inherits from the abstract **DataProvider** class and fulfills the contract by overriding the necessary members and methods

The core DNN release provides a Microsoft SQL Server implementation of the Data Provider API.

Methods in the `DataProvider` class are called by using the `Instance` method to return the concrete instance, so let's look at the method that retrieves a Host setting of `GetHostSetting`. The following code is used to call the `GetHostSetting` method:

```
DataProvider.Instance().GetHostSetting(settingName)
```

The `GetHostSetting` method is defined in the abstract `DataProvider` base class. The `Instance` method on the `DataProvider` class is used to return an instance of the actual implementation in `SqlDataProvider`.

Prior to DNN version 7.0, the `DataProvider` method was defined as follows:

```
public abstract IDataReader GetHostSetting(string settingName);
```

with the actual concrete implementation in `SqlDataProvider` being defined as follows:

```
public override IDataReader GetHostSetting(string settingName)
{
    return SqlHelper.ExecuteReader(ConnectionString, DatabaseOwner +
        ObjectQualifier + "GetHostSetting",
    settingName);
}
```

There are a couple of things to note here. First, the contract returns an interface—`IDataReader`. This is a very common practice in the Provider Model that either the parameters or the return type is an interface.

In addition, the concrete implementation shows a reference to the `SqlHelper` class. `SqlHelper` is part of the Microsoft Data Access Application Block (DAAB). Prior to version 7, DNN used this library to improve performance and reduce the amount of custom code required for data access. The DAAB is a .NET component that uses ADO.NET to call stored procedures and execute

SQL commands on Microsoft SQL Server.

In version 7 of DNN there was a significant refactoring of the Data Access Layer. Several versions ago the development team introduced some generic methods.

```
public abstract void ExecuteNonQuery(string procedureName,
    params object[] commandParameters);
public abstract IDataReader ExecuteReader(string
procedureName,
    params object[] commandParameters);
public abstract object ExecuteScalar(string procedureName,
    params object[] commandParameters);
public abstract T ExecuteScalar<T>(string procedureName,
    params object[] commandParameters);
public abstract DataSet ExecuteDataSet(string procedureName,
    params object[] commandParameters);
```

These methods allowed developers to write code in the Business Layer like the following code (without the need to create their own Data Access Layer):

```
var taskId = DataProvider.Instance.ExecuteScalar<int>("CreateTask",
    taskIdName);
```

Prior to DNN 7, it had been the practice to create a new `DataProvider` method for each new stored procedure created in the database. In DNN 7, the practice changed to use these generic methods. While the existing `DataProvider` methods needed to be retained, as they were considered part of the core API, they were refactored to use these new generic methods.

For example, the `GetHostSetting` method described earlier in this section was refactored to

```
public virtual IDataReader GetHostSetting(string settingName)
{
    return ExecuteReader("GetHostSetting", settingName);
}
```

Note that for backward compatibility the method was converted to a virtual method (rather than abstract) once the `SqlDataProvider` implementation of the method was removed.

There was a second major change made to the Data Access Layer in DNN 7. The `SqlDataProvider` methods were refactored to use `PetaPoco`—a newer, more flexible replacement for `SqlHelper`.

NOTE

Peta Poco is micro-ORM released under an open source license by Top Ten Software. For more information, see <http://www.toptensoftware.com/petapoco/>.

Data Access Layer (DAL) 2

In addition to the changes identified in the previous section, DNN 7 introduced a completely new Data Access Layer—DAL 2. The concept of the DAL 2 was twofold:

- Make it easier for developers to work with the data layer by removing the need for developers to write T-SQL
- Introduce best practices that have been identified in the last few years for working with data

After much research it was decided that the new Data Access Layer would be based on PetaPoco. As mentioned previously, PetaPoco can be thought of as a second-generation Data Access Block, but it is much more than that. Whereas the Data Access Block deals with low-level data objects—`IDataReader` and `DataSet`—PetaPoco works with Plain Old CLR Objects (POCOs).

For example, the `ExecuteReader` method in the `SqlHelper` class that is used in the Microsoft DAAB returns an `IDataReader`. This is demonstrated in the `GetTabs` method in the `SqlDataProvider` class in DNN 6.

```
public override IDataReader GetTabs(int portalId)
{
    return SqlHelper.ExecuteReader(ConnectionString,
        DatabaseOwner + ObjectQualifier + "GetTabs",
        GetNull(portalId));
}
```

Because an `IDataReader` is returned, the `GetTabs` method in `TabController` still needs to create `TabInfo` objects from the returned `IDataReader`.

```
var tabs = CBO.FillCollection<TabInfo>(Provider.GetTabs(portalID));
```

Using the DAL 2 in the Business Layer could be rewritten as shown by the following code:

```
using (IDataContext ctx = DataContext.Instance())
{
    var rep = ctx.GetRepository<TabInfo>();
    tabs = rep.Get<int>(portalId);
}
```

There is no need to write an additional Data Access Layer method to return an `IDataReader`, and there is no need to call a helper class to hydrate the tabs. The core DAL 2 classes working with PetaPoco do all the grunt work for you. For most scenarios you no longer need to write stored procedures, although there are still valid reasons to do so—for instance to improve performance.

Business Layer

Sitting on top of the Data Access Layer, the Business Layer provides the business logic for all core site activity. This layer exposes many services to the DNN core as well as to third-party extensions. These services include

- Page management
- File management
- Localization
- Caching
- Exception management
- Event logging
- Personalization
- Search
- Installation and upgrades
- Membership, roles, and profile
- Security permissions

The `JavaScriptLibraryController` class described earlier in the discussion of the Service Locator Pattern provides an excellent example of a Business Layer Controller. [Listing 7.5](#) shows the complete class listing.

[Listing 7.5](#): The JavaScriptLibraryController Class

```
public class JavaScriptLibraryController
```

```

        : ServiceLocator<IJavaScriptLibraryController,
        JavaScriptLibraryController>
        , IJavaScriptLibraryController
{
    private void ClearCache()
    {
DataCache.RemoveCache(DataCache.JavaScriptLibrariesCacheKey);
    }
    protected override Func<IJavaScriptLibraryController>
GetFactory()
    {
        return () => new JavaScriptLibraryController();
    }
    #region IJavaScriptController Implementation
    public void DeleteLibrary(JavaScriptLibrary library)
    {
DataProvider.Instance().ExecuteNonQuery("DeleteJavaScriptLibrary",
library.JavaScriptLibraryID);
        ClearCache();
    }
    public JavaScriptLibrary GetLibrary(Func<JavaScriptLibrary,
bool> predicate)
    {
        return GetLibraries().SingleOrDefault(predicate);
    }
    public IEnumerable<JavaScriptLibrary>
GetLibraries(Func<JavaScriptLibrary,
bool>
predicate)
    {
        return GetLibraries().Where(predicate);
    }
    public IEnumerable<JavaScriptLibrary> GetLibraries()
    {
        return CBO.GetCachedObject<IEnumerable<JavaScriptLibrary>>
(
            new
CacheItemArgs(DataCache.JavaScriptLibrariesCacheKey,
DataCache.JavaScriptLibrariesCacheTimeout,
DataCache.JavaScriptLibrariesCachePriority),
            c =>
CBO.FillCollection<JavaScriptLibrary>(
DataProvider.Instance().ExecuteReader("GetJavaScriptLibraries"));
    }
    public void SaveLibrary(JavaScriptLibrary library)

```



```

    {
        library.JavaScriptLibraryID =
            DataProvider.Instance().ExecuteScalar<int>
("SaveJavaScriptLibrary",
library.JavaScriptLibraryID,
                                library.PackageID,
library.LibraryName,
library.Version.ToString(3),
                                library.FileName,
library.ObjectName,
library.PreferredScriptLocation,
                                library.CDNPath);
        ClearCache();
    }
#endregion
}

```

This is a simple repository class that provides the so-called CRUD (Create, Retrieve, Update, and Delete) operations, but it demonstrates a couple of important practices.

- **Simplicity:** Where possible, the DotNetNuke API aims for simplicity. This is not always possible, and sometimes APIs need to be extended. But this API covers all aspects of manipulating the JavaScriptLibrary object with five simple methods. This is achieved by providing two very flexible Get methods that take a LINQ predicate.
- **Caching:** No single DNN install should have more than 100 JavaScript Libraries, and in general the set of JavaScript Libraries will be constant for large periods of time, so JavaScript libraries are a great candidate for caching.
- **Constants:** Note the use of constants for the caching properties. This ensures that spelling mistakes in so-called magic strings don't cause the application to misbehave.
- **ClearCache:** The class provides a private `ClearCache` method so that both the `DeleteLibrary` and `SaveLibrary` methods can ensure the cache is cleared when the collection is changed for any reason.
- **GetCachedObject:** The `GetLibraries` method uses the core

`GetCachedObject` method, which provides a thread-safe mechanism to manage the cache.

Presentation Layer

At the top of the stack is the Presentation Layer. The Presentation Layer provides the interface for users to access the application. This layer consists of the following elements:

- **Web Forms:** The main web form is `Default.aspx`. This page is the principle entry point to the application. It is responsible for dynamically loading the other elements of the Presentation Layer. `Default.aspx` is in the root installation directory.
- **Skins:** The `Default.aspx` web form loads the skin for the page based on the settings for each page and site. The base `Skin` class is in the `DotNetNuke.UI.Skins` namespace. You can find the base `Skin` class in the `/Library/UI/Skins/Skin.cs` file.
- **Panes:** The `Pane` class was introduced in DNN 5. A skin can contain multiple content panes, which form a collection of pane objects based on the design and settings for each skin, page, and site. The `Pane` class is located in the `DotNetNuke.UI.Skins` namespace. You can find the base `Pane` class in the `/Library/UI/Skins/Pane.cs` file.
- **Containers:** The pane object loads the container for each module based on the settings for each module, page, and site. The base `Container` class is located in the `DotNetNuke.UI.Containers` namespace. You can find the base `Container` class in the `/Library/UI./Containers/Container.cs` file.
- **Module controls:** All modules have at least a single control that is the user interface for the module. These controls are loaded by a container object based on the settings for each module, page, and site. The module user controls are in `.ascx` files in `/DesktopModules/[module name]`.
- **Client-side scripts:** There are several client-side JavaScript files that are used by the core user-interface framework. For example, the `/DotNetNuke/js/dnn.controls.dnnmenu.js` script file is used by the `DNNMenu` control. Custom modules can include and reference JavaScript files as well. Client-side JavaScript files that are used by the core are in the `/js` folder or in the `/resources` folder. Some skins may use client-side JavaScript, in which case the scripts are in the skin's installation directory.

Any client-side scripts used by modules are located under the module's installation directory.

- **Services Framework endpoints:** DNN 7 introduced a Services Framework based on ASP.NET Web API. This provides REST-like endpoints that can be called from client-side code. In general, these services consume or return JSON-formatted data.

When a visitor first browses to a DNN site, the web form that loads is `Default.aspx`. The code behind for that page is `Default.ascx.cs` and is found in the website's root folder. The code behind file loads the selected skin for the current page.

The skin is a custom user control that must inherit from the base `Skin` class. Prior to DNN 5 the `Skin` class was where most of the code that handled the loading of modules and containers was located. In DNN 5, the code was refactored, and the `Pane` class introduced.

The `Skin` class processes the skin file, instantiates a `Pane` object for each pane defined in the skin, and places them into a `Panes` collection. Next, the skin iterates through the list of modules that are part of the current page and passes the module to the appropriate `Pane` object. If there is more than one module in the same pane, then the pane in turn has a collection of `Modules`.

Next, each `Pane` object determines which `Container` is assigned to each module. The `Pane` instantiates a `Container` object for each module and passes the module to the appropriate `Container` object.

Skins, containers, and modules can all include their own style sheets, and each one determines if the file exists; if it does, it registers the file with the `ClientResourceManager`. The `ClientResourceManager` class ensures that any client resources (CSS files or JS files) are rendered only once in a page and are rendered in the correct order and in the correct location. See [Chapter 17](#), “Skinning,” for more information about skinning.

Namespace Overview

DNN 7 is quite a large framework, but it is organized into a coherent collection of namespaces and classes. Namespaces are a way to group classes that are related to one another. For example, all the classes that are used to manage pages (tabs) are located in the `DotNetNuke.Entities.Tabs` namespace. By grouping related classes together, developers need to use only a single `using` statement to be able to reference all the classes in the same namespace.

```
Using DotNetNuke.Entities.Tabs;
```

Most of the namespaces that are relevant to the third-party DNN developer are located in two assemblies: `DotNetNuke.dll` and `DotNetNuke.Web.dll`.

DotNetNuke.dll

Here is a brief description of the namespaces found in the main assembly `DotNetNuke.dll`:

- **DotNetNuke.Collections:** This namespace is used for the thread-safe collections (and their supporting classes) that are used in DNN: for example, `SharedList` and `SharedDictionary`.
- **DotNetNuke.Common:** The common namespace is primarily used for static helper classes like `Globals`, `CBO`, `XmlUtils`, and so on.
- **DotNetNuke.ComponentModel:** The `ComponentModel` namespace is used for the IoC Container and its supporting classes.
- **DotNetNuke.Data:** This namespace is used for all classes that make up the Data Access Layer.
- **DotNetNuke.Entities:** This namespace is used for classes that make up the core entities of DNN, e.g., `PortalInfo`, `TabInfo`.
- **DotNetNuke.ExtensionPoints:** This namespace is new in version 7.1 of DNN and provides classes that support new MEF (Managed Extensibility Framework) endpoints.
- **DotNetNuke.Framework:** This namespace provides classes that provide support for the DNN Framework itself.
- **DotNetNuke.Modules:** This namespace is the home of modules that are included as part of DNN.

- **DotNetNuke.Security:** This namespace provides classes that are involved in securing DNN; for example, roles and permissions.
- **DotNetNuke.Services:** This namespace is the largest namespace and provides a home for all the services provided by DNN. These include caching, localization, logging, and so on.
- **DotNetNuke.UI:** This namespace includes all the classes that are part of the web UI—`Skin`, `Pane`, `Container`, and some module base classes and interfaces; for example, `ModuleUserControlBase`, `IActionControl`, `ISkinControl`.

DotNetNuke.Web.dll

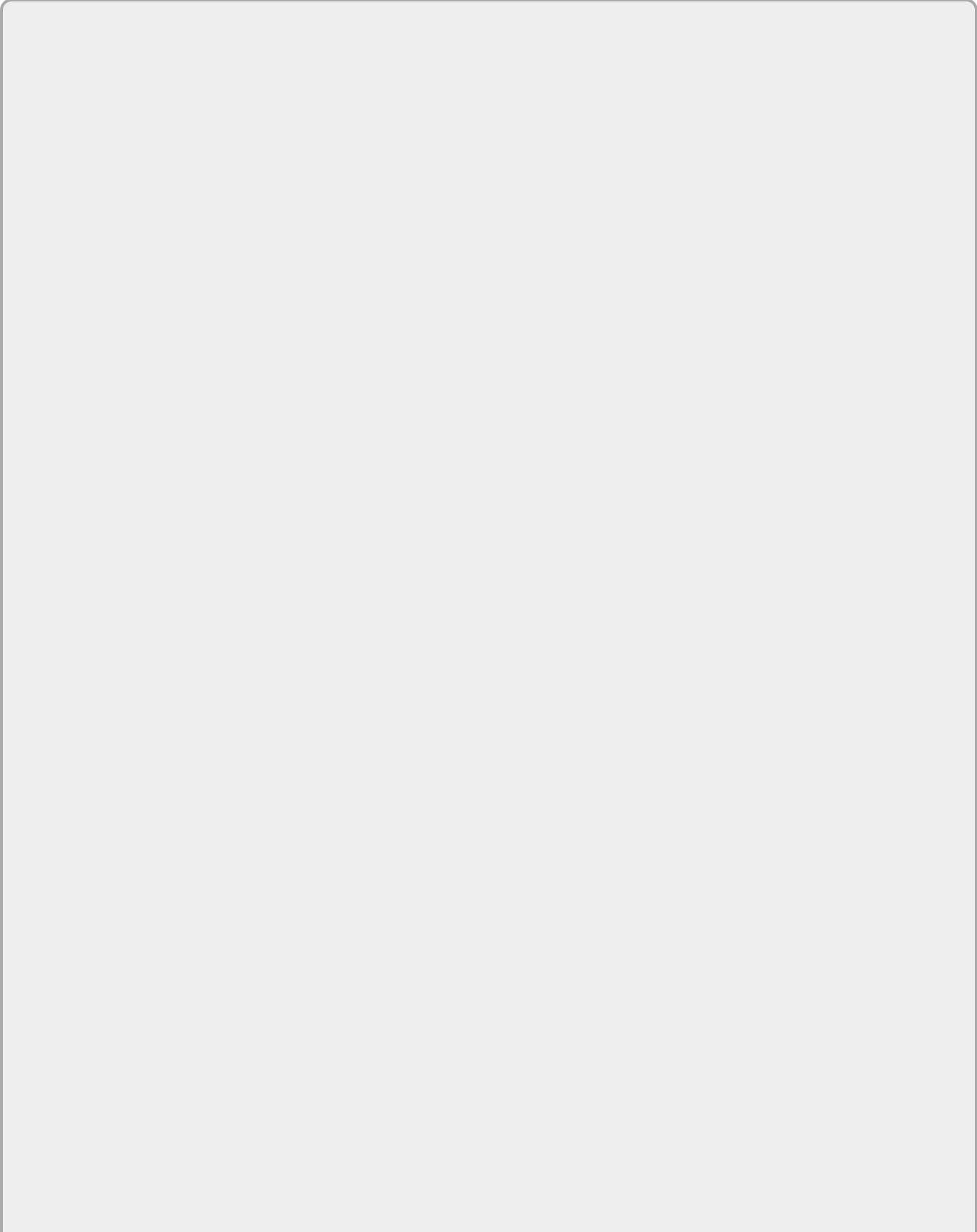
DotNetNuke.Web mainly contains classes there are used in the Presentation Layer, including a lot of web controls based on Telerik RAD Controls as well as the Services Framework classes. All namespaces in this assembly are part of the DotNetNuke.Web namespace.

- **DotNetNuke.Web.Api:** This namespace includes classes that support the new Web API–based Services Framework.
- **DotNetNuke.Web.InternalServices:** This namespace includes the internal Web API–based services used within the platform.
- **DotNetNuke.Web.Mvp:** This namespace includes classes that support DNN's implementation of the Model-View-Presenter pattern based on WebFormsMVP.
- **DotNetNuke.Web.UI:** This namespace includes web controls used by module controls. It is mainly made up of lightweight wrappers for the Telerik RAD Controls.
- **DotNetNuke.Web.Validators:** This namespace includes attribute-based validators used by the MVP implementation.

Summary

In this chapter you have reviewed the major components of the DNN Architecture as found in DNN 7. You have reviewed the common patterns and best practices as well as the different layers found in DNN. Finally you have reviewed the most commonly used namespaces that are of interest to third-party developers. In the next chapter you will look at some of the most common APIs and how they can be used.

Chapter 8
Core DNN APIs



What's in this chapter?

- Using hydration methods
- Caching data
- Logging events
- Managing exceptions
- Scheduling tasks
- Examining module interfaces

DNN provides significant capability straight out of the box. You just have to install and go. Sometimes, however, you may need to extend the base framework. DNN provides a variety of integration points from HTTP handlers to custom modules. To take full advantage of the framework, it is important to understand some of the base services and APIs provided.

This chapter examines some of the core services provided by DNN. You can use these services in your own code. As many of the services are built using the Provider model, you can also provide alternate implementations to swap out the base functionality for your own version. For example, if you want your events logged to a custom database or to the Windows Event Logs, just create your own Logging Provider.

The CBO Class

If you are fetching data from the database, then you may be interested in what the `CBO` class has to offer. `CBO` stands for Custom Business Object, and the class is designed to serialize and de-serialize objects.

The `CBO` methods can be broken down into three groups.

- Methods that are used to “create” (or hydrate) business objects from readers returned from the database
- Methods that are used to serialize a business object into XML
- Methods that are used to de-serialize XML into a business object

The first group of methods are the ones that are used most by third-party developers while the serialization and de-serialization methods are primarily used by the core extension package creator and extension installer.

Hydration Methods

[Table 8.1](#) shows the list of hydration methods provided by the `CBO` class. There are a lot of methods that you can use, and the choice of method depends primarily on what type of collection you want to use. Internal APIs that have been around since the early days of DNN are more likely to use methods that return an `ArrayList`, while APIs that have been developed in the last few years predominantly use the generic forms of the methods.

[Table 8.1](#) The CBO Hydration Methods

Method	Description
<code>CreateObject</code> (3 overloads)	Technically this method is not a hydration method. The <code>CreateObject</code> method can be used to create new business objects.
<code>FillObject</code> (4 overloads)	The <code>FillObject</code> methods are used to create a single object from a data reader. These days the two generic <code>FillObject<T></code> methods are probably used most of the time.
<code>FillCollection</code> (8 overloads)	The <code>FillCollection</code> methods are used to create a collection from a data reader. Again, the generic methods <code>FillCollection<T></code> methods are probably used most of the time.

<code>FillDictionary</code> (5 overloads)	The <code>FillDictionary</code> methods are used to create a dictionary from a data reader. You can provide the name of a property to use as the dictionary key or a default field will be used.
<code>FillQueryable</code>	The <code>FillQueryable</code> method is used to create a <code>Queryable</code> collection from a data reader.
<code>FillSortedList</code>	The <code>FillSortedList</code> method is used to create a <code>SortedList<TKey, TValue></code> collection from a data reader.

Let's look at a simple example of the use of the `FillCollection` method.

```
IDataReader reader =
DataProvider.Instance().ExecuteReader("GetTasks");
IList<Task> tasks = CBO.FillCollection<Task>(reader);
```

The `DataProvider`'s `ExecuteReader` method returns an `IDataReader` that is converted by `FillCollection` into a `List<Task>`.

The `IHydratable` Interface

The `CBO` hydration methods use reflection to map the data in the data reader. Reflection is not always the fastest way to do this mapping so `DNN` includes a mechanism to delegate the mapping to the object itself.

This is accomplished using the `IHydratable` interface. The `IHydratable` interface contains two members.

```
public interface IHydratable
{
    int KeyID { get; set; }
    void Fill(IDataReader dr);
}
```

Module developers can implement this interface in their own objects, and the `Fill` method would then do the mapping. The advantage here is that the module developer knows the structure of the data being returned and does not need to do any reflection.

The `ModuleControlInfo` class provides an example of a `Fill` method implementation.

```
public void Fill(IDataReader dr)
{
    ModuleControlID = Null.SetNullInteger(dr["ModuleControlID"]);
    FillInternal(dr);
    ModuleDefID = Null.SetNullInteger(dr["ModuleDefID"]);
    ControlTitle = Null.SetNullString(dr["ControlTitle"]);
}
```

```

        IconFile = Null.SetNullString(dr["IconFile"]);
        HelpURL = Null.SetNullString(dr["HelpUrl"]);
        ControlType = (SecurityAccessLevel) Enum.Parse(typeof
            (SecurityAccessLevel),
Null.SetNullString(dr["ControlType"]));
        ViewOrder = Null.SetNullInteger(dr["ViewOrder"]);
        SupportsPopUps = Null.SetNullBoolean(dr["SupportsPopUps"]);
        //Call the base classes fill method to populate base class
properties
        base.FillInternal(dr);
    }

```

In this case, the method maps specific columns to the relevant properties but also calls a method in the base class to map the columns that are relevant to the base class.

The `KeyID` property is used only by the `FillDictionary` method. It is used to identify the “value” to be used as the key in the Dictionary for each instance. Usually the `KeyID` implementation just returns the value of the property that is used as the primary key in the database. For example, the `KeyID` property in `ModuleControlInfo` is as follows:

```

public int KeyID
{
    get
    {
        return ModuleControlID;
    }
    set
    {
        ModuleControlID = value;
    }
}

```

Implementing this interface is quite simple and ensures that the hydration of your objects is as fast as it can be. It also ensures that the mapping is done correctly, especially when using enum-type properties.

Caching

Database access can be relatively expensive, especially if the database does not reside on the same server as the web server. While, ultimately, everything has to be retrieved and persisted to a database, you can use caching to save some data in memory for fast access.

ASP.NET provides an in-process cache, but in a cloud or web-farm scenario it is often better to use a shared out-of-process cache. For that reason, DNN uses a Provider model to implement caching.

The DataCache Class

In a similar way to the `CBO` class, the `DataCache` class works as a helper class to DNN's caching framework. [Table 8.2](#) lists the important methods of the `DataCache` class.

Table 8.2 Selected Public Methods in `DataCache`

Method	Description
<code>ClearXXXCache()</code> (16 variants)	This group of methods allows developers to clear groups of cached items. For example, <code>ClearFolderCache(portalId)</code> allows the developer to pass the current portal ID and clear all folder-related caches for that portal.
<code>GetCache()</code> (2 overloads)	<code>GetCache</code> allows a developer to retrieve a specific cached object based on its key.
<code>GetCachedData<T>()</code>	This method allows the developer to provide a callback to reload the cache if the item is not present in the cache.
<code>RemoveCache()</code>	<code>RemoveCache</code> allows a developer to clear a specific cached object from the cache based on its key.
<code>SetCache</code> (6 overloads)	<code>SetCache</code> allows a developer to save an item in the cache. The different overloads allow the developer to set a cache dependency, a priority, or an expiry time.

GetCachedData

The most important of all the methods in this class is `GetCachedData`. One of the challenges with caching is that the ASP.NET cache is not inherently

thread-safe. This means that as developers we need to ensure that multiple ASP.NET requests do not try to modify the cache at the same time.

`GetCachedData<T>` ensures that cached data is managed in a thread-safe manner. This method is so important in caching data that it also can be found in the `CBO` class—`GetCachedObject<T>`. `CBO.GetCachedObject<T>` simply wraps `GetCachedData<T>`, so any discussion of one applies to both methods.

Look an example of the method's use.

```
public IEnumerable<JavaScriptLibrary> GetLibraries()
{
    var cacheArgs = new
CacheItemArgs(DataCache.JavaScriptLibrariesCacheKey,
                DataCache.JavaScriptLibrariesCacheTimeout,
                DataCache.JavaScriptLibrariesCachePriority);
    return CBO.GetCachedObject<IEnumerable<JavaScriptLibrary>>
(cacheArgs,
    c => CBO.FillCollection<JavaScriptLibrary>
(DataProvider.Instance()
    .ExecuteReader("GetJavaScriptLibraries")));
}
```

In this case, we are using the version in `CBO`. The `GetCachedObject<T>` method takes two parameters. The first is an instance of `CacheItemArgs`. The `CacheItemArgs` class provides information about how the object is cached. [Table 8.3](#) describes the main properties of the `CacheItemArgs` class.

Table 8.3 Properties of the `CacheItemArgs` Class

Property	Description
<code>CacheCallback</code>	This property defines the <code>CacheItemRemovedCallback</code> delegate, which is called when ASP.NET removes the cached item from cache.
<code>CacheDependency</code>	This property defines the <code>DNNCacheDependency</code> property. This class wraps the ASP.NET <code>CacheDependency</code> class and identifies whether the cached object is removed if a dependent object changes (for example, a cached version of an XML file should be removed from the cache if the XML file is modified).
<code>CacheKey</code>	A unique string that identifies the object in the cache.
<code>CachePriority</code>	An enumeration that identifies the priority of the object. This is used by the ASP.NET cache to determine which

	objects to remove from the cache when resources are tight.
CacheTimeout	A multiplier that is used to determine how long the object should be cached. The multiplier is used in combination with a Host Setting (None - 0, Low - 1, Medium - 3, High - 6) to calculate for how many minutes the item is cached. The expiry time is a sliding expiry and represents the time since the object was last referenced.
Params	An array of parameters that can be used by the callback method.
ParamList	The same array converted into an <code>ArrayList</code> .

The second parameter is a delegate of type `Func<CacheItemArgs, TObject>`. In the example, the delegate is written in the form of a lambda expression. The delegate takes a `CacheItemArgs` as a parameter and returns an object of the same type of the original call to `GetCachedObject<TObject>`—in this case an `IEnumerable<JavaScriptLibrary>`.

There are a lot of examples of the use of this method in the core, and developers should take advantage of this important method when implementing caching.

Event Logging

DNN provides a rich logging API. It is designed to handle a wide variety of logging needs including exception logging, event auditing, and security logging. Like many of DNN's services, it uses the Provider model. The default logging provider saves the logs in the main DNN database, but in theory a developer could create a provider that saves the logs in the Windows Event Log, for example.

There are two major logging use cases for developers.

- Creating new logging types
- Logging an event using the logging API

Creating New Logging Types

While the logging system in DNN provides a lot of logging types, the system is extensible and allows developers to add new logging types. For example, if you are creating a module to manage a task list, then you may want to log when a user creates, updates, or deletes a task.

Adding a new log type is a two-step task. I first need to create a new `LogTypeInfo` object and save it to the database. The code to add a “CreateTask” log type is shown in the following code snippet:

```
var logController = new LogController();
var logTypeInfo = new LogTypeInfo
{
    LogTypeKey = "CreateTask",
    LogTypeFriendlyName = "Task Created",
    LogTypeDescription = "A new task was created",
    LogTypeCSSClass = "ItemCreated",
    LogTypeOwner = "DotNetNuke.Logging.EventLogType"
};
logController.AddLogType(logTypeInfo);
```

Once you add a new log type to the system, you have to configure how it behaves. This is also configurable by the administrator in the Event Viewer.

```
var thresholdType =
LogTypeConfigInfo.NotificationThresholdTimeTypes.Seconds;
var logTypeConf = new LogTypeConfigInfo
{
    LoggingIsActive = true,
    LogTypeKey = "CreateTask",
```

```

        KeepMostRecent = "100",
        NotificationThreshold = 1,
        NotificationThresholdTime = 1,
        NotificationThresholdTimeType = thresholdType,
        MailFromAddress = Null.NullString,
        MailToAddress = Null.NullString,
        LogTypePortalID = "*"
    };

```

```
LogController.Instance.AddLogTypeInfo(logTypeConf);
```

Note that both of the methods to add the type and the configuration information are found in the `LogController` class. `LogController` is the main entry point into the logging API. [Table 8.4](#) shows a list of the methods of this class.

Table 8.4 The `LogController` Class

Method	Description
<code>AddLog(LogInfo)</code>	Adds a new <code>LogInfo</code> instance to the database.
<code>AddLogTypeInfo(LogTypeInfo)</code>	Adds a new <code>LogTypeInfo</code> instance to the database.
<code>AddLogTypeInfo(LogTypeInfo)</code>	Adds a new <code>LogTypeInfo</code> instance to the database.
<code>ClearLog()</code>	Clears the log.
<code>DeleteLog(LogInfo)</code>	Deletes a specific <code>LogInfo</code> instance from the database.
<code>DeleteLogTypeInfo(LogTypeInfo)</code>	Deletes a specific <code>LogTypeInfo</code> instance from the database.
<code>DeleteLogTypeInfo(LogTypeInfo)</code>	Deletes a specific <code>LogTypeInfo</code> instance from the database.
<code>GetLogs(int, string, int, int, ref int)</code>	Retrieves a page of log records from the database.
<code>GetLogTypeInfo()</code>	Retrieves all the <code>LogTypeInfo</code> instances from the database.

<code>GetLogTypeInfoByID(int)</code>	Retrieves a single <code>LogTypeInfo</code> instance from the database.
<code>GetLogTypeInfoDictionary()</code>	Retrieves a dictionary of all the <code>LogTypeInfo</code> instances from the database. The dictionary uses the <code>LogTypeKey</code> property of the <code>LogTypeInfo</code> object as the dictionary key.
<code>PurgeLogBuffer()</code>	Purges any in-memory log records to the database.
<code>UpdateLogType(LogTypeInfo)</code>	Updates a specific <code>LogTypeInfo</code> instance in the database.
<code>UpdateLogTypeInfo(LogTypeInfo)</code>	Updates a specific <code>LogTypeInfo</code> instance in the database.

Logging an Event Using the Logging API

Now that you know how to add custom log types to the database, you can review how to use the logging API to add events. This section reviews how to log audit type events. Later, we review how to use the logging API to log exceptions.

To add a new log entry, you need to create a new `LogInfo` instance and call the `AddLog` method of `LogController`. For example:

```
var logTypeKey =
EventLogController.EventLogType.TABMODULE_DELETED.ToString();
var log = new LogInfo { LogTypeKey = logTypeKey };
log.LogProperties.Add(new LogDetailInfo("tabId", tabId.ToString()));
log.LogProperties.Add(new LogDetailInfo("moduleId",
moduleId.ToString()));
LogController.Instance.AddLog(log);
```

In this example, a `LogInfo` instance of type `TabModule Deleted` is created. Two properties are added to the `LogProperties` collection, and then the log entry is added to the database.

[Table 8.5](#) shows the properties of the `LogInfo` class.

Table 8.5 Properties of the `LogInfo` Class

Property	Description
<code>LogGUID</code>	A globally unique identifier
<code>LogTypeKey</code>	The key that represents the type (<code>LogTypeInfo</code> instance) of this log entry
<code>LogUserID</code>	The ID of the user who was responsible for this log entry
<code>LogUserName</code>	The username of the user who was responsible for this log entry
<code>LogPortalID</code>	The ID of the portal (site) where this log entry originated
<code>LogPortalName</code>	The name of the portal (site) where this log entry originated
<code>LogCreateDate</code>	The date (and time) the log entry was created
<code>LogProperties</code>	A collection of properties associated with this log entry
<code>BypassBuffering</code>	A flag that indicates whether this log entry should be persisted immediately, bypassing any buffering that may be configured
<code>LogServerName</code>	The name of the server where this log entry was created
<code>LogConfigID</code>	The ID of the configuration info (<code>LogTypeConfigInfo</code> instance) that controls the behavior of this log type

The `EventLogController` Class

While the `LogController` class provides an `AddLog` method to add a `LogInfo` instance, the `EventLogController` extends that class with a number of helper methods.

For example, if all you want to do is log that a property was changed, you can use the following code:

```
var eventLogController = new EventLogController();
var eventType = EventLogController.EventType.ADMIN_ALERT;
eventLogController.AddLog("Username", user.Username, eventType);
```

Another commonly used overload takes an object as the first parameter.

```
var eventLogController = new EventLogController();
```

```
var eventLogType = EventLogController.EventLogType.  
DESKTOPMODULE_CREATED;  
var portalSettings = PortalController.GetCurrentPortalSettings();  
var userID = UserController.GetCurrentUserInfo().UserID;  
  
eventLogController.AddLog(desktopModule, portalSettings, userID, "",  
eventLogType);
```

When this overload is used, the object that is passed in (in this case an instance of `DesktopModuleInfo`) is serialized to XML and the XML is stored in the `LogProperties`.

Exception Management

Exceptions are events. They are a special class of events, but at their most fundamental level they are events. DNN recognizes this by using the same mechanism to log handled exceptions.

DNN supports six main exception event types.

- **ModuleLoadException:** An exception type for exceptions thrown within modules
- **PageLoadException:** An exception type for exceptions thrown within pages
- **SchedulerException:** An exception type for exceptions thrown within the Scheduler
- **SearchException:** An exception type for exceptions thrown within the search engine
- **SecurityException:** An exception type for security exceptions
- **GeneralException:** All other exceptions

The Exceptions Class

DNN provides a helper class to help developers use the Exceptions API. This class provides a number of helper methods, as described in [Table 8.6](#).

Table 8.6 Helper Methods in the Exceptions Class

Method	Description
ProcessHttpException (5 overloads)	This method is used to log HTTP exceptions. These are usually 404s - Page Not Found, but maybe 500s - Internal Server Error. Unlike other exceptions, these are logged as Host Alerts.
ProcessModuleLoadException (7 overloads)	These methods are used to log exceptions that are part of the module load process. In addition to logging the exception, these methods inject a message into the page to inform the user of the exception.
ProcessPageLoadException	These methods are used to log exceptions that

(2 overloads)	occur during the page load process. As with module exceptions, these methods also inject a message into the page to inform the user of the exception.
LogException (5 overloads)	These methods are used to simply log an exception.
ProcessShedulerException	This method logs a scheduler exception.
LogSearchException	This method logs a search exception.

The following snippet shows an example of how these methods can be used:

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);
    try
    {
        ... Code that causes an exception to be thrown
    }
    catch (Exception exc)
    {
        Exceptions.ProcessModuleLoadException(this, exc);
    }
}
```

Scheduler

DNN provides a Scheduler component. This is a mechanism that enables administrators to schedule tasks to run at defined intervals. As with most services, the DNN Scheduler is implemented using the Provider pattern, so it can easily be replaced without modifying core code.

From a developer's perspective there is an API that allows developers to create their own scheduled task. This is a fairly simple process.

However, before looking at the Scheduler API, it's important to understand which types of tasks are suitable for the Scheduler.

The Scheduler is run under the context of the web application. Therefore, it is prone to the same types of application recycles as a web application. For example, every time the application is recycled by recycling the worker process, the Scheduler will stop running. Tasks run by the Scheduler therefore do not run 24 hours a day, 7 days a week—they are executed according to a defined schedule—but they can be triggered only when the worker process is alive. Therefore, you cannot specify that a task should run every night at midnight. It is not possible in the web environment to meet this type of use case. You can, however, specify how often a task is run by defining the execution frequency for each task. The execution frequency is defined as every x minutes/hours/days.

To create a scheduled task, you must create a class that inherits from `DotNetNuke.Services.Scheduling.SchedulerClient`. This class must provide a constructor and a `DoWork` method. An example of a scheduled task's `DoWork` method is shown in the following code:

```
public override void DoWork()
{
    try
    {
        string str = CachingProvider.Instance().PurgeCache();
        ScheduleHistoryItem.Succeeded = true; //REQUIRED
        ScheduleHistoryItem.AddLogNote(str);
    }
    catch (Exception exc) //REQUIRED
    {
        ScheduleHistoryItem.Succeeded = false; //REQUIRED
        ScheduleHistoryItem.AddLogNote(string.Format("Purging cache task
failed: {0}.",
            exc.ToString()));
        //notification that we have errored
    }
}
```

```
    Errored(ref exc); //REQUIRED
    //log the exception
    Exceptions.Exceptions.LogException(exc); //OPTIONAL
}
}
```

This scheduled task is part of the core and purges (clears) the cache. Many scheduled tasks in the core are marked with “//Required” comments and “Optional” comments. This indicates the pieces of code that should be implemented in the scheduled task for consistency, and you can use this as a guide when writing your own tasks.

For example, the task is built using the try/catch pattern and the value of the `Succeeded` property of the `ScheduleHistoryItem` property of the base class is set to true or false.

Module Interfaces

Interfaces provide a way for the core to communicate with the module. You have already seen an example of one of the interfaces that Module developers can implement—`IHydratable`. `IHydratable` can be implemented by the module's entity classes to allow the module developer more control over how the entity is hydrated with data from the database.

Although there are others, the main interfaces that module developers can use are as follows:

- `IModuleControl`: All module controls must implement this interface (usually through inheriting from a base class).
- `IActionable`: This interface provides the ability for module developers to define action menu items.
- `IPortable`: This interface provides support for exporting and importing module content.
- `IModuleCommunicator`: This interface is used to support sending messages to other modules.
- `IModuleListener`: This interface is the complement of the previous interface and is used to support receiving messages from other modules.
- `IUpgradeable`: This interface is used to identify code that should be executed when the module is upgraded (or installed for the first time).
- `ISearchable`: This legacy interface was used to allow modules to participate in DNN's legacy Search capabilities. It has been replaced by a new Search API, which is discussed in [Chapter 11](#).

The chapters on module development later in this book explore the interfaces in more detail, so just a few of the interfaces are briefly explored here.

`IModuleControl`

The `IModuleControl` interface has five members.

```
public interface IModuleControl
{
    Control Control { get; }
    string ControlPath { get; }
    string ControlName { get; }
    string LocalResourceFile { get; set; }
```



```

ModuleInstanceContext ModuleContext { get; }
}

```

[Table 8.7](#) provides more detail about the members of the interface.

[Table 8.7](#) Members of the IModuleControl Interface

Method	Description
Control	This property returns the underlying control—most of the time this control is a <code>UserControl</code> , but it can (rarely) be a <code>WebControl</code> .
ControlPath	The path to the control. For a <code>UserControl</code> this is something like <code>DesktopModules/Admin/SQL/SQL.ascx</code> . For a <code>WebControl</code> it is the fully qualified class name for the control.
ControlName	The name of the control.
LocalResourceFile	The path to the associated default local resource file. Most of the time the name of the local resource file will be the same as the module control itself, with an additional <code>.resx</code> extension. It will also be stored in the <code>App_LocalResources</code> folder for the module.
ModuleContext	This property provides access to the detailed context of the module.

In addition, DNN provides a number of base classes that implement this interface.

- **PortalModuleBase:** This is the original class that implements the interface. It inherits from `UserControl`. In addition to the five members, it provides a number of helper properties that expose specific properties of the `ModuleContext` property.
- **ModuleUserControlBase:** This newer class also inherits from `UserControl` and provides a cleaner implementation. It provides only one additional member—the `LocalizeString` method.
- **ModuleControlBase:** This class inherits from `Control` and can be used to create fully compiled server controls.
- **CachedModuleControl:** This class inherits from `Literal` and is used by the DNN core to model “cached” module controls.

IActionable

When DNN is in Edit mode, every module has an Action menu that provides a list of actions. Many of these actions are provided by the DNN core. The `IActionable` interface allows module developers to add their own custom actions to this menu.

```
public interface IActionable
{
    ModuleActionCollection ModuleActions { get; }
}
```

The interface has a single member—the `ModuleActions` property. This property returns a list of custom actions defined for the module.

The following code shows an example usage as implemented in the Banners module. `ModuleActions` is a read-only method, so you only need to provide the `get` function. The first step is to create a new collection to hold the custom actions. Then you use the collection's `Add` method to create a new action item in the collection. Finally, you return the new collection.

```
public ModuleActionCollection ModuleActions
{
    get
    {
        var Actions = new ModuleActionCollection();
        Actions.Add(GetNextActionID(),
            Localization.GetString(ModuleActionType.AddContent,
LocalResourceFile),
            ModuleActionType.AddContent,
            "",
            "",
            EditUrl(),
            false,
            SecurityAccessLevel.Edit,
            true,
            false);
        return Actions;
    }
}
```

The `ModuleAction` class is the heart of this API. [Table 8.8](#) shows the members available in this class. Each item in the module action menu is represented by a single `ModuleAction` instance.

[Table 8.8](#) Members of the ModuleAction Class

--	--

Method	Description
Actions	Contains the collection of module action items that can be used to form hierarchical menu structures. Every skin object that inherits from <code>ActionBase</code> may choose how to render the menu based on the capability to support hierarchical items. For example, the default skin object supports submenus, whereas the <code>DropDownActions</code> skin object supports only a flat menu structure.
ID	Every module action for a given module instance must contain a unique Id. The <code>PortalModuleBase</code> class defines the <code>GetNextActionId</code> method, which can be used to generate unique module action IDs.
CommandName	Distinguishes which module action triggered an action event. DNN includes a number of standard <code>ModuleActionTypes</code> that provide access to standard functionality. Custom module actions can use their own string to identify commands recognized by the module.
CommandArgument	Provides additional information during action event processing. For example, the DNN core uses <code>CommandArgument</code> to pass the module ID for common commands like <code>DeleteModule.Action</code> .
Title	Sets the text that is displayed in the module action menu.
Icon	Name of the Icon file to use for the module action item.
Url	When set, this property allows a menu item to redirect the user to another web page.
ClientScript	JavaScript to run during the menu's <code>click</code> event in the browser. If the <code>ClientScript</code> property is present, it is called prior to the postback occurring. If the <code>ClientScript</code> returns false, the postback is canceled.
UseActionEvent	Causes the site to raise an action event on the server and notify any registered event listeners. If <code>UseActionEvent</code> is false, the site handles the event but does not raise the event back to any event listeners. The following <code>CommandNames</code> prevent the action event from firing: <code>ModuleHelp</code> , <code>OnlineHelp</code> , <code>ModuleSettings</code> , <code>DeleteModule</code> , <code>PrintModule</code> , <code>ClearCache</code> , <code>MovePane</code> , <code>MoveTop</code> , <code>MoveUp</code> ,

	MoveDown, and MoveBottom.
Secure	Determines the required security level of the user. If the current user does not have the necessary permissions, the module action is not displayed.
Visible	If set to false, the module action will not be displayed. This property enables you to control the visibility of a module action based on custom business logic.
NewWindow	Forces an action to open the associated URL in a new window. This property is not used if UseActionEvent is true or if the following CommandNames are used: ModuleHelp,Online Help,ModuleSettings, orPrintModule.

IPortable

DNN provides the ability for modules to be able to import and export their content. The `IPortable` interface defines the methods that a module developer must implement to enable this.

```
public interface IPortable
{
    string ExportModule(int ModuleID);
    void ImportModule(int ModuleID, string Content, string Version, int
UserID);
}
```

This is a pretty simple interface to implement. `ExportModule` allows modules to export their content by serializing it as XML, and `ImportModule` allows modules to import content serialized as XML. (Actually, as `ExportModule` returns a string and `ImportModule` accepts a string, it is up to the developer to determine how to serialize the content, but as the core uses XML for serializing the module settings, it is better to play along and use XML.)

The following code is the `IPortable` implementation of the HTML module:

```
public string ExportModule(int moduleId)
{
    string xml = "";
    ModuleInfo module = ModuleController.Instance.GetModule(moduleId,
Null.NullInteger, true);
    int workflowID = GetWorkflow(moduleId, module.TabID,
module.PortalID).Value;
    HtmlTextInfo content = GetTopHtmlText(moduleId, true, workflowID);
    if ((content != null))
```

```

    {
        xml += "<htmltext>";
        xml += "<content>" +
XmlUtils.XmlEncode(TokeniseLinks(content.Content,
                                module.PortalID)) + "</content>";
        xml += "</htmltext>";
    }
    return xml;
}
public void ImportModule(int ModuleID, string Content, string
Version, int UserId)
{
    ModuleInfo module = ModuleController.Instance.GetModule(ModuleID,
                                                            Null.NullInteger, true);
    var workflowStateController = new WorkflowStateController();
    int workflowID = GetWorkflow(ModuleID, module.TabID,
module.PortalID).Value;
    XmlNode xml = Globals.GetContent(Content, "htmltext");
    var htmlContent = new HtmlTextInfo();
    htmlContent.ModuleID = ModuleID;
    // convert Version to System.Version
    var objVersion = new Version(Version);
    if (objVersion >= new Version(5, 1, 0))
    {
        // current module content
        htmlContent.Content =
DeTokeniseLinks(xml.SelectSingleNode("content")
                .InnerText, module.PortalID);
    }
    else
    {
        // legacy module content
        htmlContent.Content =
DeTokeniseLinks(xml.SelectSingleNode("desktophtml")
                .InnerText, module.PortalID);
    }
    htmlContent.WorkflowID = workflowID;
    htmlContent.StateID = workflowStateController
                .GetFirstWorkflowStateID(workflowID);
    // import
    UpdateHtmlText(htmlContent,
GetMaximumVersionHistory(module.PortalID));
}

```

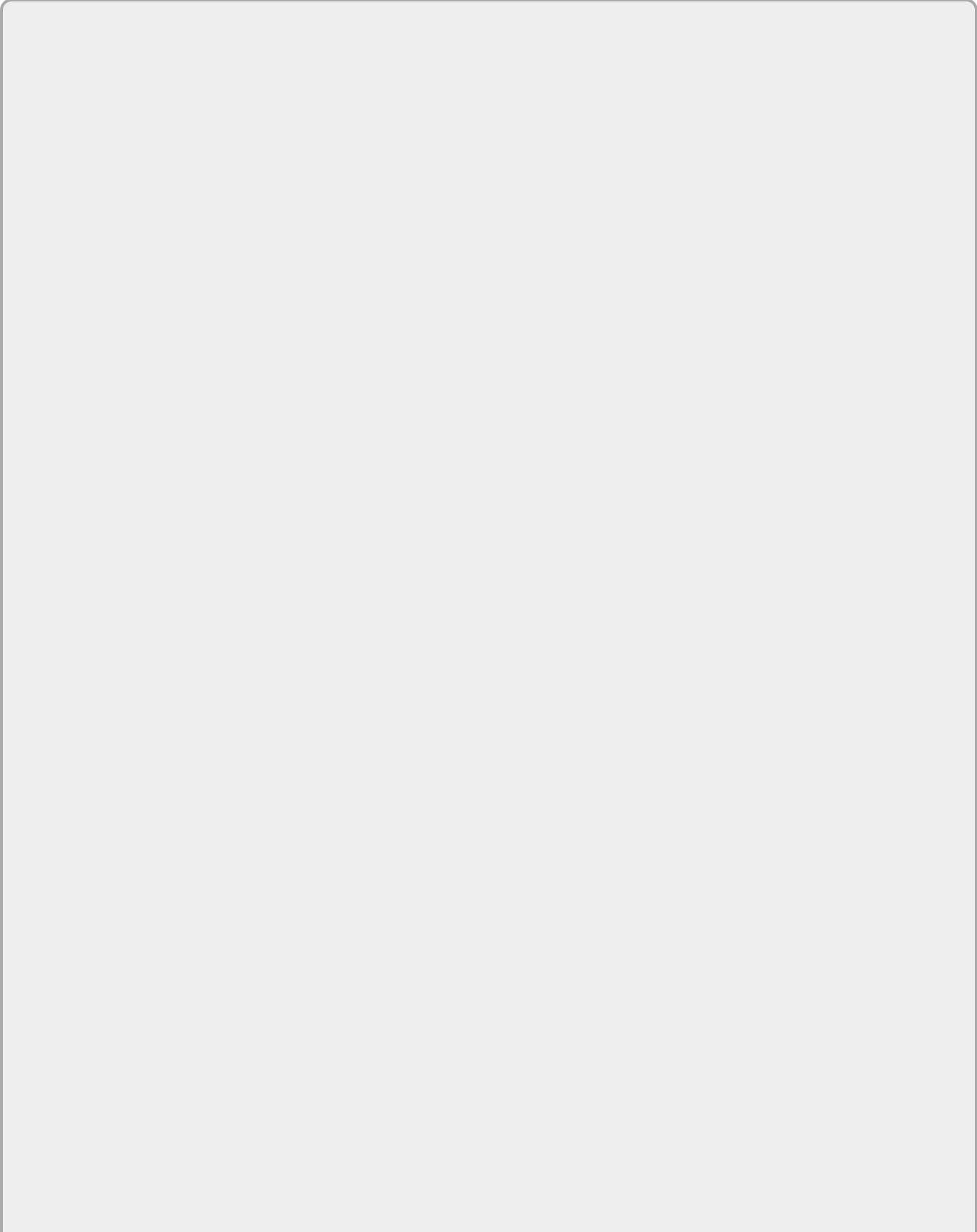
The HTML module's data model is simple, so the resulting implementation is also quite simple. If your data model is more complex, then the implementation will also be more complex. The `SerializeObject` and `DeserializeObject` methods of the CBO class can help with the serialization and de-serialization.

Summary

This chapter examined many of the core APIs that provide the true power behind DNN. By leveraging common APIs, you can extend DNN in almost any direction. You can replace core functions or just add a custom module. The core APIs are what makes it all possible.

Chapter 9

Membership Security



What You Will Learn in This Chapter

- Creating a membership provider
- Creating authentication providers
- Managing membership security

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 9](#) download and individually named according to the names throughout the chapter.

DNN has a mature, robust membership-management system that has worked well for many years and supported many use cases and extension scenarios. However, as with all code, time moves on and new requirements surface that the existing codebase struggles to support. From a security perspective, this is particularly critical, with changes in the security landscape necessitating changes in DNN's membership system. These changes led to a large-scale review of the Membership Security surface in DNN and the 7.1.0 release, which resulted in considerable change. In this chapter, you'll look at the most common extension scenarios as well as the recent changes in the Membership Security model.

DNN Membership Overview

Since its inception as the IBuySpy site, DNN has contained rich member, role, and profile support. In the earliest days this involved its own API, custom tables, and stored procedures. However, when Microsoft announced their intention to introduce its own controls, SQL Server entities, and API to manage these areas in ASP.NET 2.0, DNN decided to move from their custom solution to Microsoft's version. By doing so, DNN benefited in many ways, from reducing code to leveraging documentation to working seamlessly in shared hosting environments.

The analysis at the time showed that some elements could easily be reused, others required some work, and others were not appropriate.

In ASP.NET 3.5, several new security enhancements expand on these services in three distinct ways.

- **Login and user controls:** ASP.NET 2.0 introduced new controls for login and user management. However, as DNN already had similar controls in place, these were skipped for the initial DNN 3.0 code. Later, DNN introduced authentication providers as a first-class extension to the platform that allows login controls to be replaced and added to (this is covered in more detail later in this chapter).
- **ASP.NET Web Configuration:** Each ASP.NET 3.5 application can be accessed through a special set of administrative pages that enable an authorized user to create new users, assign users to roles, and store user information. Again, as DNN had these items in place, DNN skipped this integration.
- **Membership/Roles Provider:** The membership feature creates a link between the front-end features (login controls and user-management site) and the persistence mechanism. A Membership Provider encapsulates all the data access code that is required to store and retrieve users and roles. Thanks to the Provider model, this component can easily be replaced with a provider that supports your particular data source. This was the area of greatest integration. Our analysis showed that we would have to substitute the `MemberRole` application ID with a DNN portal ID.

Although many elements (membership, roles, and profiles) use the Provider model and can be swapped out, in practice the default implementations work for most people. In this chapter, you'll look at the areas that are most

commonly changed—the Membership Provider and authentication providers.

Membership Provider

As previously mentioned, DNN does not use ASP.NET's Membership Provider functionality directly. Instead, it uses its own `AspNetMembershipProvider` (found in the `DotNetNuke.Security.Membership` namespace), which handles writing data to the relevant Microsoft tables as well as DNN's custom tables. In this way DNN can leverage much of Microsoft's work, while adding its own custom logic where required. Although the default DNN implementation meets the majority of a site's needs, it also acts as an extension point for sites with their own custom requirements.

Prior to the DNN 7.1.0 release, the full list of DNN abstract provider membership methods and properties that could be overwritten with a custom implementation was as follows:

- `MinPasswordLength`: Minimum length required for a password
- `MinNonAlphanumericCharacters`: Number of non-alphanumeric characters required in a user's password
- `PasswordFormat`: How the passwords are stored in the data store; the options are Clear, Hashed, or Encrypted
- `PasswordStrengthRegularExpression`: A regular expression each password is passed through to verify it meets additional criteria
- `RequiresQuestionAndAnswer`: Determines if users are required to have a question and answer for accessing their password
- `ChangePassword`: Changes a user's password
- `ChangePasswordQuestionAndAnswer`: Changes a user's password question and answer
- `CreateUser`: Creates a single user
- `DeleteUser`: Deletes a single user
- `GetPassword`: Returns the password of a user
- `GetUser`: Returns a single user
- `GetUserByUserName`: Returns a single user by the username
- `GetUserMembership`: Returns all the membership-specific information for a single user

- `GetUsers`: Returns a list of users
- `GetUsersByEmail`: Returns a list of users by email address
- `GetUsersByUsername`: Returns a list of users by username
- `GetUsersByProfileProperty`: Returns a list of users who meet criteria by various profile properties
- `ResetPassword`: Resets a user's password
- `UnLockUser`: Unlocks user accounts so they can log in to the site
- `UpdateUser`: Updates a single user
- `UserLogin`: Authenticates a single user
- `MaxInvalidPasswordAttempts`: Property that contains the maximum number of invalid login attempts before locking a user out

In DNN 7.1.0, the following methods/properties were added as virtual methods to allow better support for hashed passwords, as well as to support vanity URLs and unique display names:

- `GetUserByVanityUrl`: Returns a single user by its vanity URL
- `GetUserByDisplayName`: Returns a single user by the display name
- `ResetAndChangePassword`: Resets and changes a user's password
- `ChangeUsername`: Changes a user's username
- `AddUserPortal`: Adds a user to another site

All of these new functions are virtual functions; if you want to implement your own UI that doesn't utilize these functions (such as a custom authentication provider), your provider does not need to provide implementations. If they are accessible via the UI, an exception will be thrown.

On the surface there seems to be nothing surprising here; it's simply a set of new methods that allow for enhancements in the Membership Security area. One of these methods is `ResetAndChangePassword`, and it's important because of a change in DNN's default security model.

In versions of DNN prior to 7.1.0, the default password format was encrypted. This meant that all passwords stored in the system used Triple-DES encryption (with a custom salt per user) to store users' passwords. This can

be seen in the standard definition in `web.config`, as shown in [Listing 9.1](#).

[Listing 9.1](#): Pre-DNN 7.1.0 Membership Configuration

```
<add name="AspNetSqlMembershipProvider"
type="System.Web.Security.SqlMembershipProvider"
connectionStringName="SiteSqlServer"
enablePasswordRetrieval="false"
enablePasswordReset="true" requiresQuestionAndAnswer="false"
minRequiredPasswordLength="7"
minRequiredNonalphanumericCharacters="0"
requiresUniqueEmail="false" passwordFormat="Encrypted"
applicationName="DotNetNuke" description="Stores and retrieves
membership data from the local Microsoft SQL Server database"/>
```

DNN 7.1.0 changed the password format setting for new installs to be `Hashed`, as shown in [Listing 9.2](#).

[Listing 9.2](#): DNN 7.1.0 Membership Configuration

```
<add name="AspNetSqlMembershipProvider"
type="System.Web.Security.SqlMembershipProvider"
connectionStringName="SiteSqlServer"
enablePasswordRetrieval="false"
enablePasswordReset="true" requiresQuestionAndAnswer="false"
minRequiredPasswordLength="7"
minRequiredNonalphanumericCharacters="0"
requiresUniqueEmail="false" passwordFormat="Hashed"
applicationName="DotNetNuke" description="Stores and retrieves
membership data from the local Microsoft SQL Server database"/>
```

This change was a response to a series of attacks against poorly secured user passwords in other (non-DNN) sites, such as the PlayStation Network, Gawker, LinkedIn, and Yahoo. In many cases, those sites used encryption and assumed that it would be sufficient to protect users. However, recent advances in a number of areas including using cloud computing resources and graphical processing units (GPUs) to decrypt passwords from stolen databases meant that hashing became the minimum default.

When changing to hashing we found that a number of DNN user functions were not supported well by Microsoft's `MembershipProvider`. One example of this is the ability of the administrator to change a user's password. As hashing

is a one-way operation, an administrator cannot retrieve a user's password. However, the user's password was needed to change the password, so in the absence of this, we added `ResetAndChangePassword` to work around this limitation. This leverages the API to change the password to a new random one, which is then used to change to the desired new password.

Authentication Providers

Authentication providers were added as an extension type shortly after the current membership structure was added. Unlike most other providers, their configuration is not set in `web.config`, but rather uses the Authentication table in DNN. This table contains details of the authentication provider type, such as whether it is enabled and controls for the login, logoff, and settings. The folder structure of the provider looks something similar to [Figure 9.1](#).

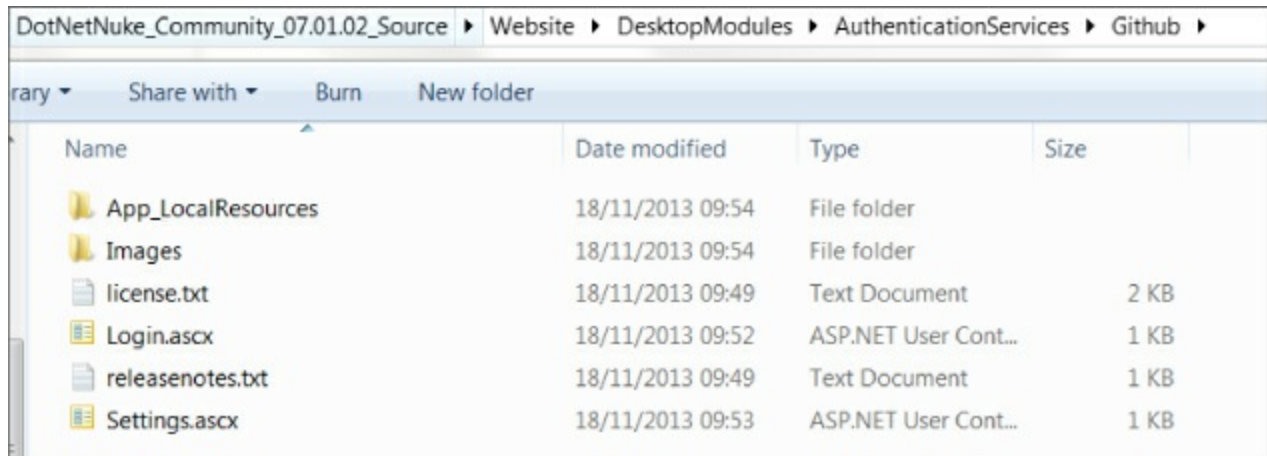


Figure 9.1

DNN ships with a pre-configured forms-based authentication provider, and the extension model makes it easy to add other custom authentication providers.

Although the design of authentication providers means you aren't tied to any particular authentication scheme (indeed the page at http://www.dnnsoftware.com/wiki/page/Authentication_Providers has many provider links that cover a broad range of third-party systems), one particular area of growth that we have seen is providers that connect to OAuth endpoints.

As the web grew, more and more sites relied on distributed services and cloud computing, along with the increased popularity of integrating these third-party services (aka “mash-ups”). In this case, a site might integrate with a third-party site such as Flickr to display photos in a gallery, or perhaps connect to a cloud device such as OneDrive to host content files.

The problem is that, in order for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing user passwords to someone else—often the same passwords used for

online banking and other site—it also provides these applications with unlimited access to do as they wish. They can do anything, including changing the passwords and locking the users out.

OAuth provides a standard method for users to grant third-party access to their resources without sharing their passwords. It also provides a way to grant limited access (in scope, duration, and so on). At the time of writing, over 60 OAuth service providers can be found at

http://en.wikipedia.org/wiki/OAuth#List_of_notable_OAuth_service_provi with many more out there not listed. (Many of these service providers implement OAuth for many products. Google supports OAuth 2.0 as the recommend authentication mechanism for all its APIs.)

Due to the popularity of OAuth-integrated systems, DNN has a number of utility classes that further simplify the creation of OAuth-based authentication providers. You should review the

`DotNetNuke.Services.Authentication.OAuth` namespace, which has a lot of code that can reduce what you'll need to write when writing a custom OAuth provider.

As an example of this, you'll use some of these utilities classes to write a provider for the popular source code repository GitHub. For the sake of brevity, we've described only a few of these classes. The full source code copy of this provider can be found at <https://github.com/cathalconnolly/DNN-GitHub-Authentication>.

The first and most important class is `OAuthClientBase`. To use it, you just have to create a custom class and inherit from this abstract class, as shown in [Listing 9.3](#).

[Listing 9.3](#): GitHub Provider Base Class

```
public class GithubClient : OAuthClientBase
{
    public GithubClient(int portalId, AuthMode mode)
        : base(portalId, mode, "Github")
    {
        TokenEndpoint = new
Uri("https://github.com/login/oauth/access_token");
        AuthorizationEndpoint =
            new Uri("https://github.com/login/oauth/authorize");
        MeGraphEndpoint = new Uri("https://api.github.com/user");
    }
}
```

```

        AuthTokenName = "GithubUserToken";
        OAuthVersion = "2.0";

        LoadTokenCookie(String.Empty);
    }
}

```

You use this to set up the custom URLs to generate the OAuth tokens and return the authenticated user graph, which you can then later use to create a DNN user. It's also important at this point to set the `AuthTokenName` to a value that ends in “UserToken”—we'll discuss the reason behind this later.

To map the user graph to a DNN user, you have to set up a class to capture the serialized data returned from the OAuth service. In this case, you inherit from `UserData` and use `DataMember` attributes to map the JSON data to them. You can then use these to populate the fields the `UserData` class expects, as shown in [Listing 9.4](#).

[Listing 9.4](#): GitHub Provider UserData Class

```

[DataContract]
public class GithubUserData : UserData
{
    #region Overrides
    public override string Locale
    {
        get { return Location; }
        set { }
    }
    public override string ProfileImage
    {
        get { return AvatarUrl; }
        set { }
    }
    public override string DisplayName
    {
        get { return GitName; }
        set { }
    }
    #endregion

    [DataMember(Name = "avatar_url")]
    private string AvatarUrl { get; set; }
    [DataMember(Name = "location")]
    private string Location { get; set; }
    [DataMember(Name = "name")]

```

```
private string GitName { get; set; }  
}
```

Now that you have the structure in place for an OAuth provider, you need to go ahead and create the login control. A login control normally has to inherit from `AuthenticationLoginBase`, which handles actions common to logging in (such as deciding if the user is authenticating as part of registering as a new user or is authenticated as an existing user). Again, you can leverage the utility classes to reduce the amount of boilerplate code required for an OAuth provider. See [Listing 9.5](#).

[Listing 9.5](#): GitHub Provider Login Control

```
public partial class Login : OAuthLoginBase  
{  
    protected override string AuthSystemApplicationName  
    {  
        get { return "Github"; }  
    }  
    public override bool SupportsRegistration  
    {  
        get { return true; }  
    }  
    protected override UserData GetCurrentUser()  
    {  
        return OAuthClient.GetCurrentUser<GithubUserData>();  
    }  
    protected override void OnInit(EventArgs e)  
    {  
        base.OnInit(e);  
        loginButton.Click += loginButton_Click;  
        registerButton.Click += loginButton_Click;  
        OAuthClient = new GithubClient(PortalId, Mode);  
        loginItem.Visible = (Mode == AuthMode.Login);  
        registerItem.Visible = (Mode == AuthMode.Register);  
    }  
    private void loginButton_Click(object sender, EventArgs e)  
    {  
        OAuthClient.Authorize();  
    }  
}
```

In this case, the login class inherits from `OAuthLoginBase`. You then provide implementations of its methods to set whether you support registration and

which class to map user graph data to. In addition, you identify the authentication system name.

Now that you've created the login class, you need to create a settings control. For a standard OAuth-based provider, you again have a base class that you inherit from—in this case `OAuthSettingsBase`. The `OAuthSettingsBase` class uses the `AuthSystemApplicationName` value to load the API key and API secret and set whether the key is enabled. It displays those values to the users, allowing them to edit/update. See [Listing 9.6](#).

[Listing 9.6](#): GitHub Provider Settings Control

```
public partial class Settings : OAuthSettingsBase
{
    protected override string AuthSystemApplicationName
    {
        get { return "Github"; }
    }
}
```



To ensure that the settings can be displayed and edited, you need to ensure that the settings control has a `propertyeditorcontrol` with a fixed name, as shown here:

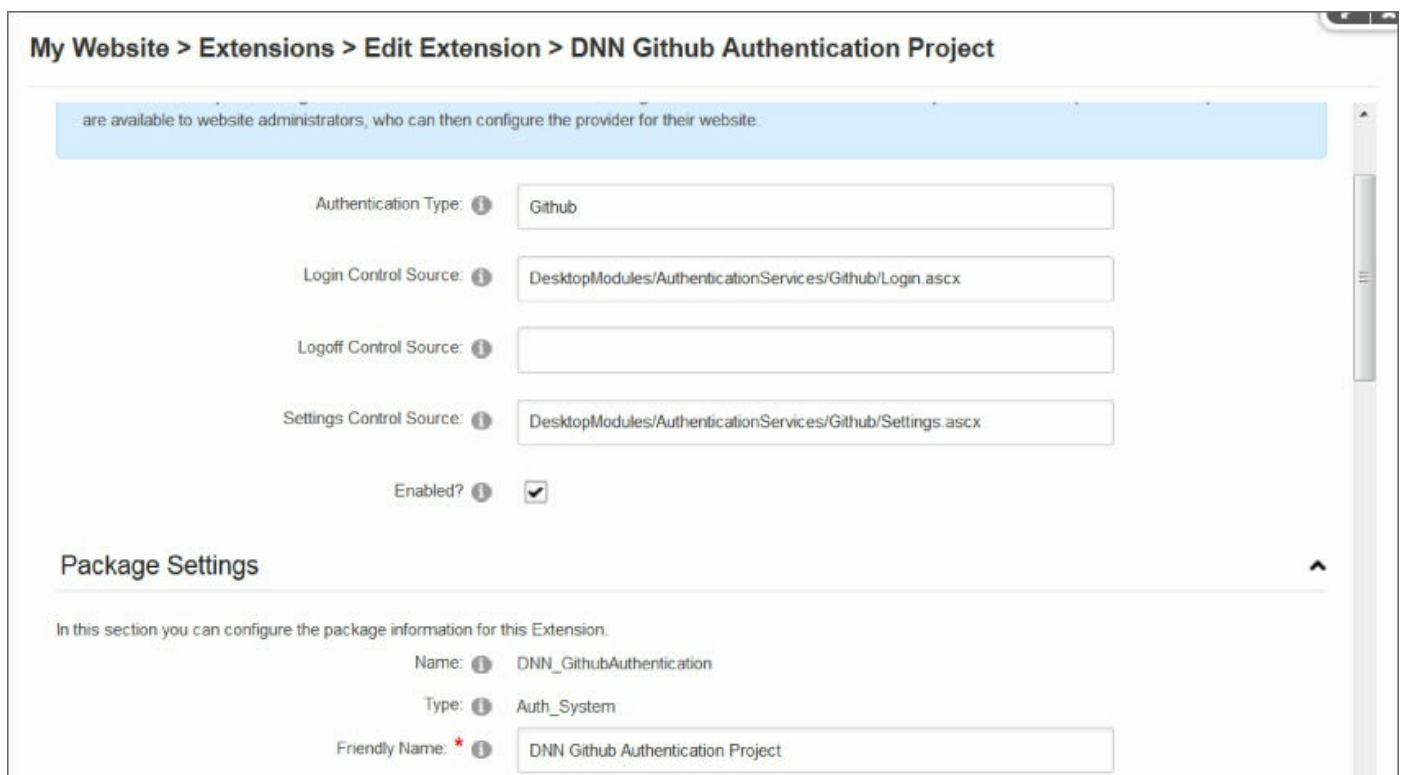
```
<dnn:propertyeditorcontrol id="SettingsEditor" runat="Server"
    helpstyle-cssclass="dnnFormHelpContent dnnClear"
    labelstyle-cssclass="SubHead"
    editmode="Edit"
    SortMode="SortOrderAttribute"
/>
```



NOTE


Although authentication providers support login, logoff, and settings pages, for an OAuth-based provider you can use configuration to remove the need for a logoff page. If you recall earlier in the `GitHubClient` class, you set the `AuthTokenName` property to `GithubUserToken`. This is important as the base logoff functionality in DNN contains code that loops over the cookie collection and will expire any cookie that ends with “UserToken.” By leveraging this convention, you can avoid the need to write a custom logoff control to simply expire the `GithubUserToken`.


Now you simply package the extension using the standard DNN manifest file structure. To use it, you install it via the Host  Extensions menu. Once installed, go to Host  Extensions and expand Authentication System. Click the pencil icon beside GitHub and ensure Enabled is selected. See [Figure 9.2](#).





My Website > Extensions > Edit Extension > DNN Github Authentication Project


are available to website administrators, who can then configure the provider for their website.


Authentication Type:  Github

Login Control Source:  DesktopModules/AuthenticationServices/Github/Login.ascx


Logoff Control Source: 


Settings Control Source:  DesktopModules/AuthenticationServices/Github/Settings.ascx

Enabled? 

Package Settings 

In this section you can configure the package information for this Extension.

Name:  DNN_GithubAuthentication

Type:  Auth_System



Friendly Name: *  DNN Github Authentication Project

Figure 9.2

Now that you've enabled it at the host level, you can configure any site that requires it. To do so, go to Admin  Extensions, click the pencil icon, and provide the APP ID and APP Secret you got when you registered the

application with the OAuth source. See [Figure 9.3](#).

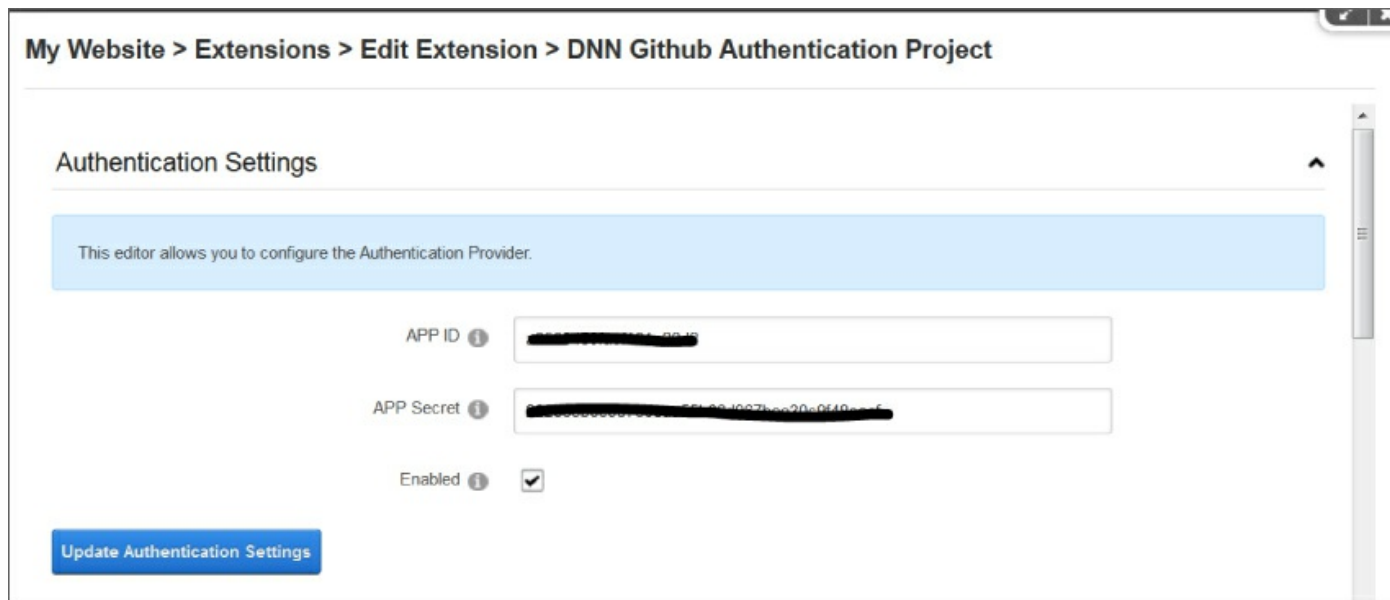


Figure 9.3

When you log in to the DNN site now, you'll see a new button to authenticate with GitHub. See [Figure 9.4](#).

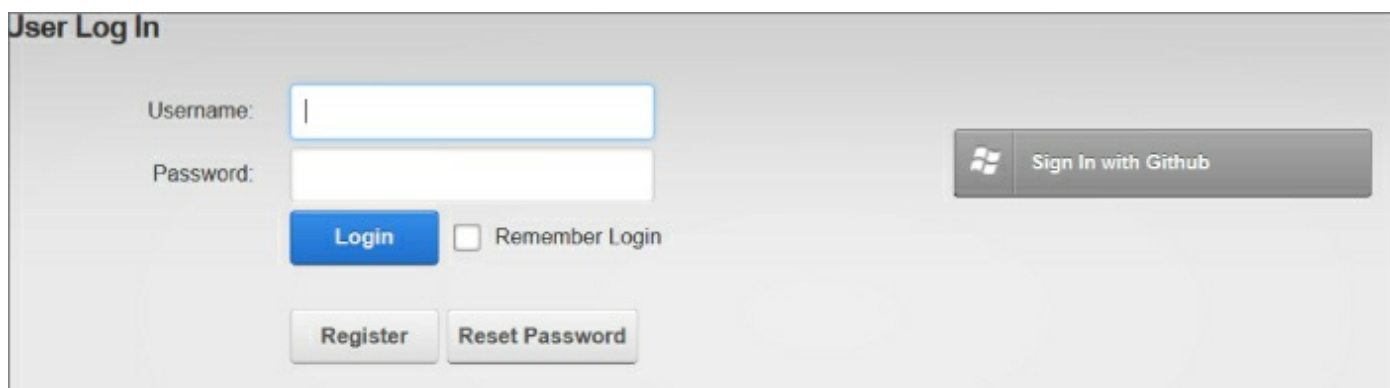
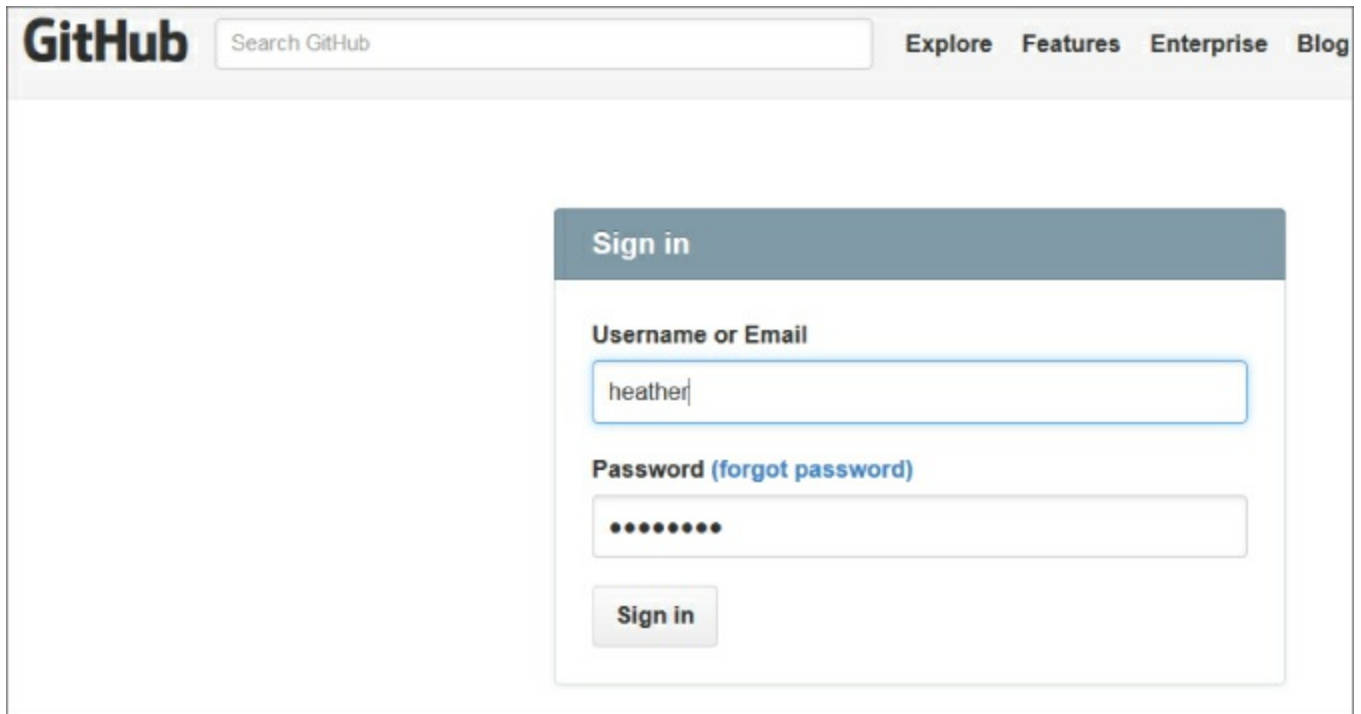


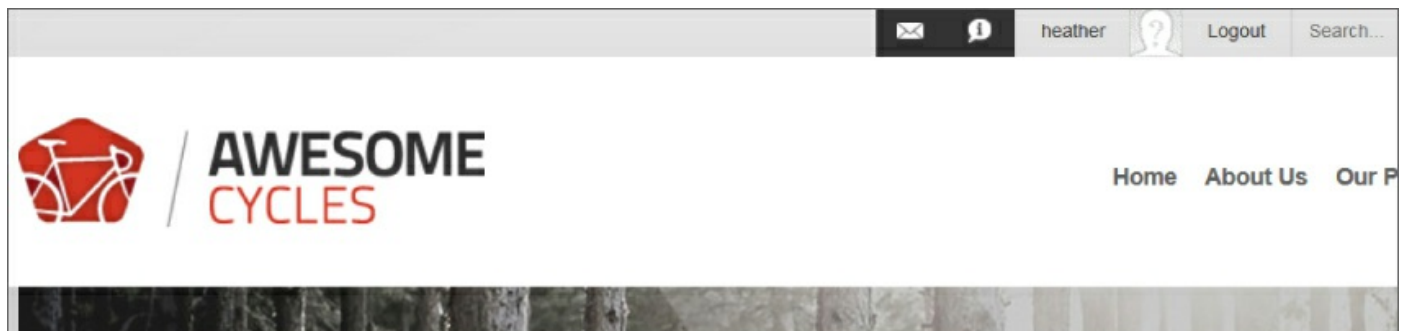
Figure 9.4

Clicking the button redirects to GitHub, where you can provide a valid user. See [Figure 9.5](#).



[Figure 9.5](#)

Once the user is authenticated, GitHub will redirect him to the application, at which point the provider registers the user. See [Figure 9.6](#).




[Figure 9.6](#)

Membership Management Enhancements

Along with any other technical implementation, security needs to evolve to address changing needs. This tends to be the emergence of new types of threats, but another important case involves technical changes that reduce or nullify the security of previously secure areas. The most apparent of these in recent memory is the change in processing power (and the ability to delegate to larger engines such as cloud services), as well as the use of cheap graphical processing units (GPUs). As you've already seen, DNN's using hashed passwords is a response to this issue, but as part of the security team review, other enhancements were deemed necessary to adjust to the changing landscape. All of the following changes were added in the DNN 7.1.0 release.

One of the largest security risks related to users' passwords are the users themselves. Users often choose short or easily guessable passwords. With the 7.1.0 release, DNN added some functionality to try to address this situation.

Log in as a superuser (such as host) and go to Host  Host Settings. Click the Advanced Settings page and then scroll down and click Membership Management. You'll find a number of settings that affect various member-related activities. See [Figure 9.7](#).



Setting	Value
Reset link timeout (minutes)	60
Enable password history?	<input checked="" type="checkbox"/>
Number of passwords to store	5
Enable password banned list	<input checked="" type="checkbox"/>
Enable password strength check	<input checked="" type="checkbox"/>
Enable IP address checking	<input type="checkbox"/>

[Figure 9.7](#)

The first of these is Reset Link Timeout. As hashing is a one-way operation—and once it's stored a user's hashed password, it cannot be retrieved (such as to send out in a password reminder)—DNN had to make changes to the platform to support password resets via reset links. Rather than have two systems—one that emailed passwords (via SMTP in cleartext) for sites that used encryption and one that sent out password reset links for sites that used hashing—the product team made the decision to only use password reset

links as they are more secure. Now, when a user initiates an action that used to send out a password (such as when registering or needing a password reminder), an email is sent with a password reset link. This link is unique to the user, can be used only once, and has an expiration to ensure that it is valid for only a short period of time. If someone were to access your email, reset links that have been used or expired can no longer be used.

The first setting on this page controls the duration and defaults to 1 hour (60 minutes). It's important to note that this value applies only to actions the user initiates. When an administrator/host creates a new account or resets a user's password, those links expire after 24 hours. This longer period allows users a better opportunity to respond to the mail by clicking the reset link. If the link expires, users can request a new one (which will then be valid for the new Reset Link Timeout duration that's specified under Membership Management). The larger timeout for admin/host-initiated actions improves the chances that the users will respond in time to the original email.



NOTE

If you happen to be running a site that uses a language pack that doesn't have the updated entries, or if you create a site-specific version of the `GlobalResources.resx` file, you might need to update the relevant keys for your site. To do so, open the


`App_GlobalResources\GlobalResources.resx` file and look for the `EMAIL_PASSWORD_REMINDER_BODY` and `EMAIL_USER_REGISTRATION_PUBLIC_BODY` keys. Edit the file to use the new `PasswordResetToken` value, as shown in [Listing 9.7](#).

[Listing 9.7](#): Example of Correctly Formatted Password Resource File

```
Link to reset password: http://[Portal:URL]/default.aspx?
ctl>PasswordReset&
resetToken=[Membership>PasswordResetToken]
If you do not know or cannot remember your password, please go to
http://[Portal:URL]/default.aspx?ctl>PasswordReset&resetToken=
[Membership>PasswordResetToken] to reset it.
```

The second setting, Enable Password History, was added in 7.1.0 to allow sites to record users' previous passwords. When users try to change their passwords, DNN will create a hash of the new password and compare it to previous passwords. If a match is found, users cannot reuse this recently used password. This setting allows site administrators to decide whether to enable this functionality. It works in concert with the next setting, which is called Number of Passwords to Store. This defaults to five, which means that users can reuse passwords after they have cycled through five previous unique ones. Although five is a sensible default, some more sensitive sites may want to increase this.

NOTE

Password history entries are created only when users change their passwords (via either a change password or reset password function), and not when the users are initially created, either by themselves (when registering) or by a site administrator (via the Admin  Users screen).

The next setting, called Enable Password Banned List, was also added in 7.1.0. Since its introduction in DNN 3.0, `web.config` has contained `minRequiredPasswordLength` (the minimum password length with a default value of seven) and `minRequiredNonalphanumericCharacters` (minimum number of non-alphanumeric characters with a default value of zero). In addition, as DNN implements the Microsoft Membership Provider, a third-property was available, although it was not used in the default configuration. This property is `passwordStrengthRegularExpression`, and it allows a site to set a .NET regular expression to evaluate password complexity. These properties offered a certain amount of security, but it recently became apparent that they were not enough. A number of high-profile sites such as the PlayStation Network, Gawker, LinkedIn, and Yahoo suffered from security breaches where the hackers were able to steal user information. Although some of these did not follow best practices, such as using password salt values, some of them were using hashing with salt values and believed that their users' passwords were reasonably safe.

Historically, hackers have been able to “crack” salted password hashes by creating a hash of a password and salt and comparing it to the user's password ciphertext. If the two matched, the hacker knew the user's password and could then try to use it to log into a website. If they have the user's username, they may try the username/password combination on sites such as Amazon, PayPal, and so on. As many users re use the same credentials on multiple sites, a hacker may gain access to the new site and use that access to steal money/order products/impersonate the user.

The computational power required to crack a single user's password used to be substantial (typically available only to governments or major academic institutions). However, with the advent of cloud computing resources and the ability to use cheap graphical processing units (GPUs) in parallel, the ability to crack hashed passwords is now available to a much wider set of people.

In addition, hackers have generated pre-computed tables of common terms hashed against common algorithms. These “rainbow tables” allow for brute-force dictionary attacks that can crack commonly used passwords easily. Taking all of this into account, DNN decided to introduce support for “banned common passwords” in DNN 7.1.0.

To generate a list of common passwords, the DNN platform team examined the cracked passwords list from attacks on Gawker, LinkedIn, and Yahoo. These entries were merged with the list of passwords from Conficker, a virus “worm” that infected millions of computers and was the largest worm infection since 2003. The team then filtered out entries shorter than six characters and finally ordered them by how common they were. During installation of 7.1.0, approximately 250 of these are added as a host-level list that will be used when the Enable Password Banned List option is enabled. As a list, hosts can add/edit/delete any entry they like. In addition, empty placeholder lists with the name `BannedPasswords<portalid>` (for example, `BannedPasswords0` for site 0) are created for each site. Additional banned passwords can be added to these, and they will be merged with the host list when a user attempts to register for a particular site. This allows site administrators to add banned passwords for one site that do not apply to another, which is particularly useful when you have sites that use different locales, as different languages will usually have different common passwords.



NOTE

If the “banned password” list is enabled, DNN will no longer allow users to use their usernames as their passwords.

The next setting, called Enable Password Strength Check, differs from previous settings in that it tries to encourage good practice rather than enforce it. It displays a rich control when users try to set their passwords (such as during registration or password reset), rather than the simple textbox. As the user enters their proposed password, a “traffic light” indicator will indicate whether the password is poor (red), somewhat secure (yellow), or very secure (green). Text is also shown so the user can see how the password is rated. If they hover over the password control, a tooltip will appear that details how this password is scored. This scoring system uses a system of rules, some configurable and some that apply best practices.

These rules are as follows:

- When the password contains one or more lowercase characters, it receives one point.
- When the password contains one or more uppercase characters, it receives one point.
- When the password contains one or more numeric characters, it receives one point.
- When the password contains special characters (equal to or greater than the value set by the `minRequiredNonalphanumericCharacters` property in `web.config`), it receives one point.
- When the password is greater than or equal to the length set by the `minRequiredPasswordLength` property in `web.config`, it receives one point.
- When the password is greater than or equal to the length plus three characters set by the `minRequiredPasswordLength` property in `web.config`, it receives one point. For example, if the `minRequiredPasswordLength` property is set to 7, the password must be 10 or more characters to score this point.

The final checkbox allows host users to enable IP address checking. This check occurs when the admin or host attempts to log in. If the IP address is

blocked or the IP is not allowed (this can be individual IPs or ranges of IP addresses), the user cannot login.

If you enable IP blocking, all users will still be able to access the site as IP blocking contains a single allow entry (“*”) that tells DNN to allow all IP address ranges. This is intended to ensure that site administrators do not accidentally lock themselves out.

IP blocking is intended for scenarios where a network has a public IP address (such as 82.24.96.98) and a private IP address (such as 192.168.27.0/24) and you want to ensure that only the users on the private IP address range can log in. In this case, you would access the site from an internal address (192.168.27.0/24) and then set 82.24.96.98 as a deny (you can now remove the *). This configuration will mean that admin/host users can log in only when they're on the 192.168.27.0/24 private network.

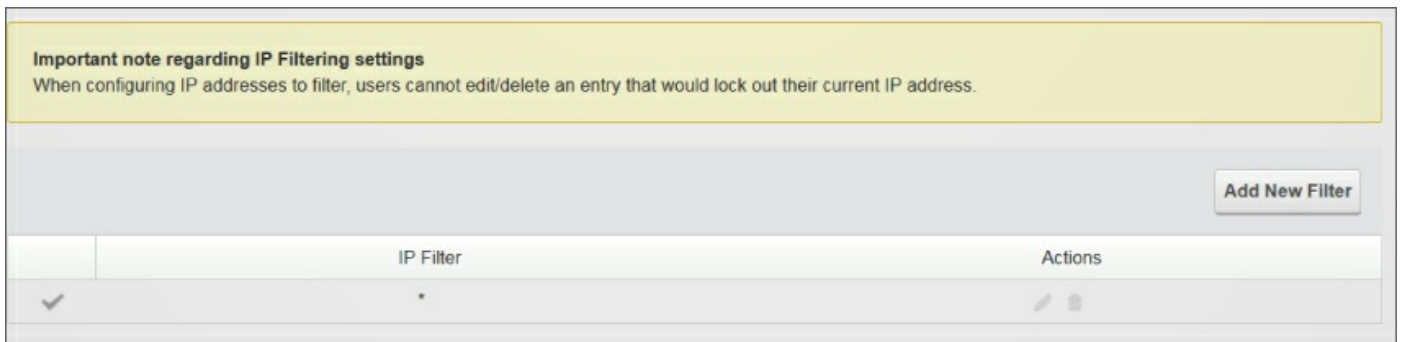
There is no current way to block an IP address you are accessing (such as 192.168.27.0/24), as that would stop you from logging in and changing the settings. (This differs from the IP restrictions in IIS, which do allow you to block your current IP address.)



NOTE

*It also supports allowing only certain public IP addresses. For example, if you accessed your site via your IP (82.24.96.98), you could set an allow rule for that IP address and then delete the *. Now you can log in as admin/host only via 82.24.96.98. No other public or private addresses or ranges ranges can log in.*

To configure this setting, enable IP address checking. Then browse to Host [Host Settings](#) [Advanced Settings](#) [Login IP Filters](#) and configure your required settings. By default, all IP addresses are allowed (represented by the use of the asterisk). As you add rules, DNN will determine whether the IP addresses you are currently on would be blocked and, if so, will not allow that rule. See [Figure 9.8](#).



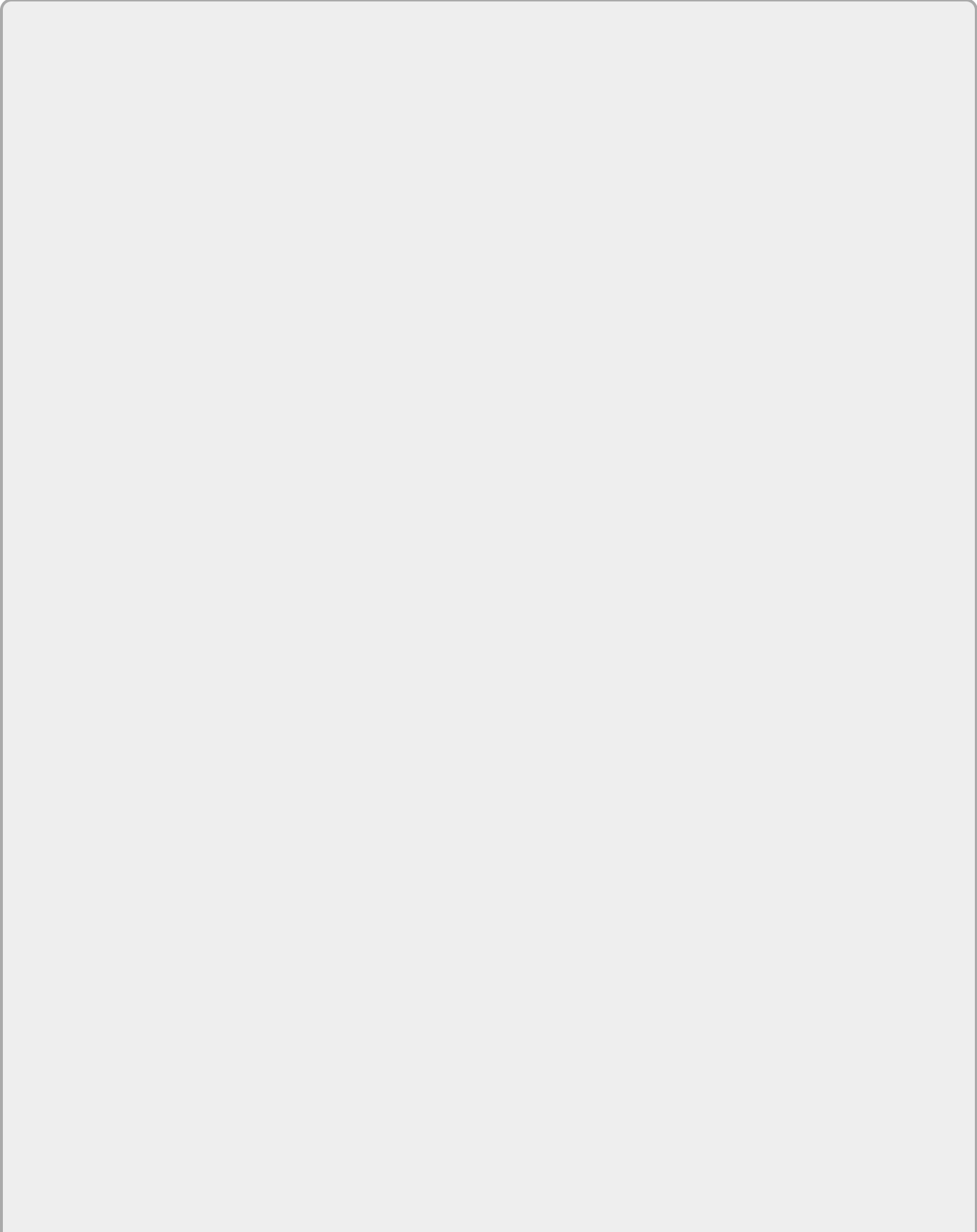
[Figure 9.8](#)

Summary

This chapter provided some background regarding the major changes in DNN's Membership Security model, as well as illustrated the most common security extension points. These details will help ensure that your DNN sites are protected against potential security issues, as well as allow you to extend and customize DNN's Membership Security model for any custom needs you have.

Chapter 10

Localization



What You Will Learn in This Chapter

- Configuring different languages
- Using the localization API in your own modules
- Managing static resources

The World Wide Web is an international network that must accommodate users from hundreds of cultures, speaking many different languages. This poses a significant challenge for every web application that is targeted at a worldwide audience. To address this challenge, DNN provides a built-in localization framework that addresses many of the issues required to make DNN usable by a global audience. This framework is built to take advantage of the ASP.NET localization features while tackling some of its shortcomings.

This chapter covers how DNN detects languages and locales and then provides instruction for using the API to implement localization in your modules. You will work with resource files (`*.resx`) to store strings that are used to localize strings of text directly in the HTML of your modules and with method calls from code-behind. You also learn about the token replacement engine and a technique for localizing images using dynamically selected CSS stylesheets.

DNN builds on the localization API in the .NET Framework to make the functionality easier to implement and manage in core and third-party modules. DNN provides localization for static text in module user interfaces and system messages and also features the ability to localize website content stored in modules dynamically. Combining these two capabilities, DNN is equipped to serve the needs of almost any international website.

Locales in DNN

A *locale* is the combination of a country code and a language code. In DNN, you will see the words locale, culture, and language used as synonyms. For the sake of accurate translations, it is important to use both a language code and a country code to perform localization. Many languages are spoken in more than one country, and dialects may differ from country to country. For example, a banking website that operates in the United States and Great Britain that needs to display “check” or “cheque” changes its interface language depending on whether the user's locale code is for English in the United States (`en-US`) or Great Britain (`en-GB`). This is the RFC 1766 standard.

Locale Detection

In order to support localization throughout the application, DNN has its own locale detection process, which is a bit more advanced than what is available in standard ASP.NET. Locale detection is important because DNN deals with both anonymous and authenticated users, and it acknowledges user preferences, either through browser settings or through user profile preferences in DNN.

Even though major refactoring was done in DNN 6 and DNN 7, locale detection has changed little since the early releases of DNN version 3. Understanding the sequence of how locales are detected can help in troubleshooting and designing localization in your modules. The `OnInit` event sets the current culture of the thread using the `GetPageLocale` method in `DotNetNuke.Services.Localization.Localization` by running through a sequence of detection starting with the language parameter of the request and finishing with the default language of `en-US` as stored in the `DotNetNuke.Services.Localization.Localization.SystemLocale` constant if no other setting is found. The exact order of detection is defined as this:

1. Querystring parameter (language)
2. Cookie value (name = language)
3. User profile (for authenticated users)
4. Browser preference
5. Site default locale
6. System locale

In this list, an item higher on the list takes precedence over an item lower on the list. For instance, if a user is authenticated but there is also an explicit culture code present in the querystring, the culture code in the querystring is used.

During steps 1 through 4 of the locale detection process, DNN will first try for an exact match of language code and country code. If an exact match is not found, it will try to find a language code match. This part is essential with languages like Spanish, which is spoken in many countries with slightly different dictionaries. If a site is designed in the Spanish language for Mexico (`es-MX`), a website visitor from Bolivia (`es-BO`) would be able to make much more sense of the Mexican Spanish dictionary than the default United States English (`en-US`).

The real utility of this process is that by the time the page begins its load process, a valid culture that is supported by the installation has been intentionally set. With DNN as a base framework, module developers can implement interface and dynamic content localization without having to manage any of the particulars of detecting and managing locales. Instead, the work is already done, and developers simply use the selected locale made available to every page, module, and skin.

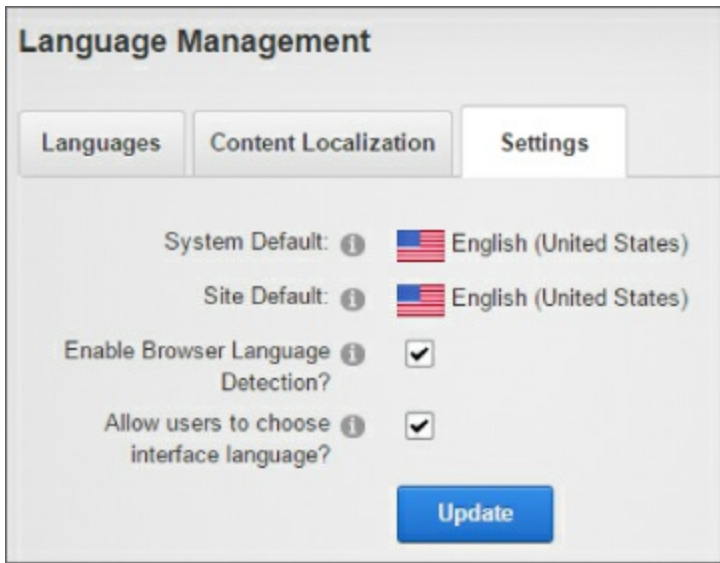
Sometimes DNN cannot set the locale in this manner. Most notably this happens when code is executed by the Task Scheduler. Since this happens on a background thread, with no HTTP context available, DNN is not able to set the locale. As a good practice, if you need your module to run in different cultures, you should always test your code using more than one culture.

Content Localization

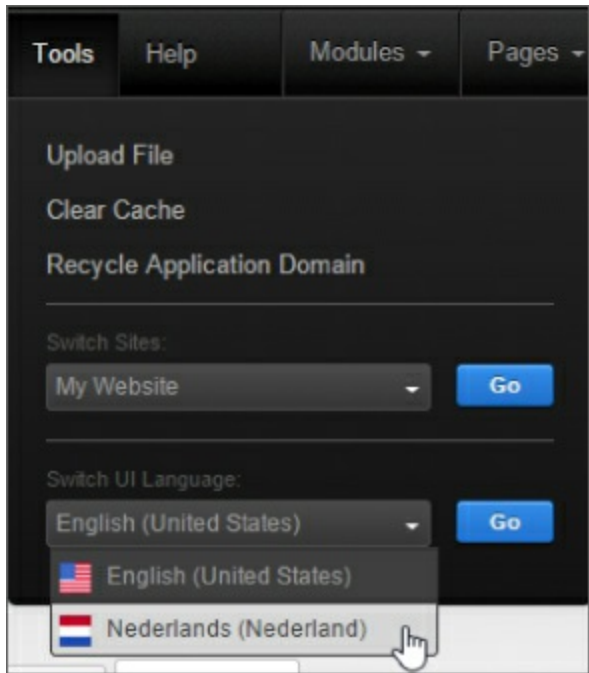
Since version 5.3, DNN also supports localization of site contents. This feature has impact on which locale is available to the module running on a page. Standard ASP.NET features two different culture settings:

`CurrentCulture` and `CurrentUICulture` (in `System.Globalization.CultureInfo`). In DNN, usually the values of these culture settings are equal; however, under content localization, they *can* be different. There is a setting available for this in the Language Management module (Admin → Languages). The setting is named “Allow Users to Choose Interface Language?” This setting is especially useful when a content editor is required to manage content for a language he does not understand.

[Figure 10.1](#) shows the setting; the default is unchecked. Once it is turned on, a new drop-down menu becomes available in the Tools menu, allowing the end user to select a different UI language. The benefit of this setting is that if the user is editing the site for a language he does not understand, he can show the user interface of the site in one language and store contents in another one. [Figure 10.2](#) shows the drop-down menu for this feature.



[Figure 10.1](#)



[Figure 10.2](#)

For a module developer, this setting is important, as it means a distinction needs to be made between `CurrentCulture` and `CurrentUICulture`. The first

should be used when displaying content, the latter for displaying UI elements.

Resource Files

To align closely to the ASP.NET localization implementation, DNN uses the Windows Resource Files (RESX) format to store translations. This file format uses XML tags to store key/value pairs of string values.

Here's an example format of a resource file's data elements shown with the Data Provider form fields on the Host Settings page:

```
. . .
  <data name="plDataProvider.Text">
    <value>Data Provider:</value>
  </data>
  <data name="plDataProvider.Help">
    <value>The provider name which is identified as the default
data
    provider in the web.config file</value>
  </data>
. . .
```

The corresponding markup in the .ascx file looks like this:

```
. . .
  <dnn:label id="plDataProvider" controlname="lblDataProvider"
  runat="server"/>
. . .
```

This is the markup for the DNN Label control. This control displays both a label and help text, accessible through an “i” icon. The control uses the `[controlname].Text` value to render the label text and the `[controlname].Help` value to render the help text.

Each name attribute is referred to as a resource key. Notice that the two resource keys in this example have extensions. DNN uses a number of extensions that help identify translations more easily. Resource keys should always use an extension, otherwise they will not be recognized.

- `.Text`: Used for the text properties of controls (used as the default if not included in resource key in code)
- `.Help`: Used for help text of the DNN Label control
- `.Header`: Used for the HeaderText properties of DataGrid columns
- `.EditText`: Used for the EditText properties of DataGrids
- `.ErrorMessage`: Used for the ErrorMessage property of Validator controls

Whereas `.Text` relates directly to the `Text` property of an ASP.NET control, the other extensions are there to instruct DNN on how to use the string for other controls.

There is one other special type of resource key, which is used by DNN to automatically localize titles of module controls. The special key has the format `ControlTitle_[ModuleControlKey].Text`, where `[ModuleControlKey]` is the control key with which the control is registered in the module definition. In the special case of the default view control, which has an empty key, the resource key would equal `ControlTitle_.Text`. However, due to the way DNN works, the title of the default view control can always be edited by the end user. Only modules that are placed on Admin pages will use this control key to define a value for the module title.

There are three types of resource files in DNN: Application Resources, Local Resources, and Global Resources. These are defined in the following sections.

Application Resources

Application Resources translations are shared throughout many controls in DNN. Application Resources files are the storage area for generic translations. For example, to localize the words “True” and “False,” you would store the translations in the Application Resources files. Other examples of Application Resources are Yes, No, Submit, and Continue.

Application Resources are stored in the `App_GlobalResources` directory, which is directly under the DNN root installation directory. The filename for the system locale `en-US` for Application Resources is `SharedResources.resx`. The filename convention for other locales is `SharedResources.[locale].resx`. For example, the German language Application Resources file for Germany would be named `SharedResources.de-DE.resx`.

Global Resources

Global Resources translations are for localizing strings from components that do not have Local Resource files, are not necessarily shared translations, and don't fit in the first two categories. Because all Local Resources are associated with a user control or a page, there is no place to store translations for components. For this reason, there's this other category for resource files. For example, almost all emails that are sent out by the DNN core framework work with templates that are stored in Global Resources. The keys for these

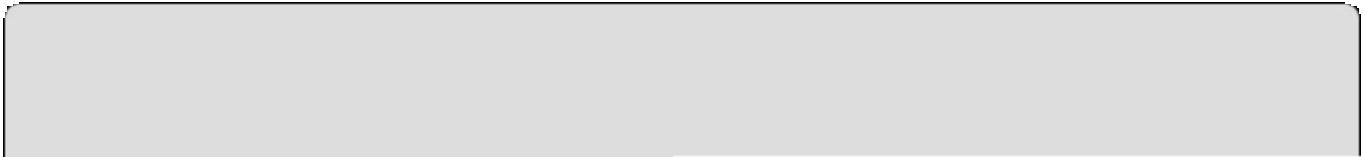
templates are usually prefixed with `EMAIL_`.

Global Resources are stored in the same directory as Application Resources (`/App_Resources`). The filename for the system default locale is `/App_Resources/GlobalResources.resx`. For locales other than the system default, the naming convention is as follows:

`/App_Resources/GlobalResources.[locale].resx`.

Local Resources

Local Resources translations are usually unique to a user control. For example, to localize the HTML module's user control, you would store the translations in a Local Resources file that lives in a child directory beneath the HTML module's directory.



NOTE

If a module has static translations that are generic in nature (such as True and False), the translations should be gathered from Application Resources.

Local Resources files are stored in a directory named `App_LocalResources`. Each directory that contains localized user controls must have a directory named `App_LocalResources`. The filename for the System Locale follows this naming convention:

```
[control_directory]/App_LocalResources/[user_control_file_name].resx
```

An example, for the `en-US` locale in the resource file for the HTML module would be as follows:

```
HTML/App_LocalResources/HtmlModule.ascx.resx
```

The filename for other locales follows this naming convention:

```
[control_directory]/App_LocalResources/[control_files_name].  
[locale].resx
```

A module will often use the same resource strings in multiple controls. In order to make housekeeping of these strings easier, a special file is available for such resources: the shared resources file. The contents of this file will always be available to all controls that belong to the module. The file must be named `SharedResources.resx`, and if a file with that name exists, it will automatically be picked up by DNN.

Resource File Handling

DNN uses an extensive fallback scheme to read from resource files, which allows for a very granular control of resources; however, it can also cause some headaches, as it is sometimes not clear from which file a specific resource value is loaded. The fallback scheme is reflected in the naming conventions of resource files. DNN will try to read a resource value first from the most specific file, and if not found, it will keep trying until the least specific file is reached. Specificity is defined not only by the language (culture code) but also by application scope: Site (portal), Application (host), and System. In order to accommodate fallback from site to system, DNN uses the

following file formats:

- **Site specific:** [ResourceFileName].Portal-[PortalId].resx
- **Application specific:** [ResourceFileName].Host.resx
- **System specific:** [ResourceFileName].resx

In real-life situations this type of fallback is very useful, as it allows site administrators to change resource values without running the risk of the changed values to be overwritten with an application upgrade.

[Figure 10.3](#) shows the built-in Language Editor, where the host user can choose at which level he wants to edit resources (mode selector). For site administrators, only the Site option will be available.

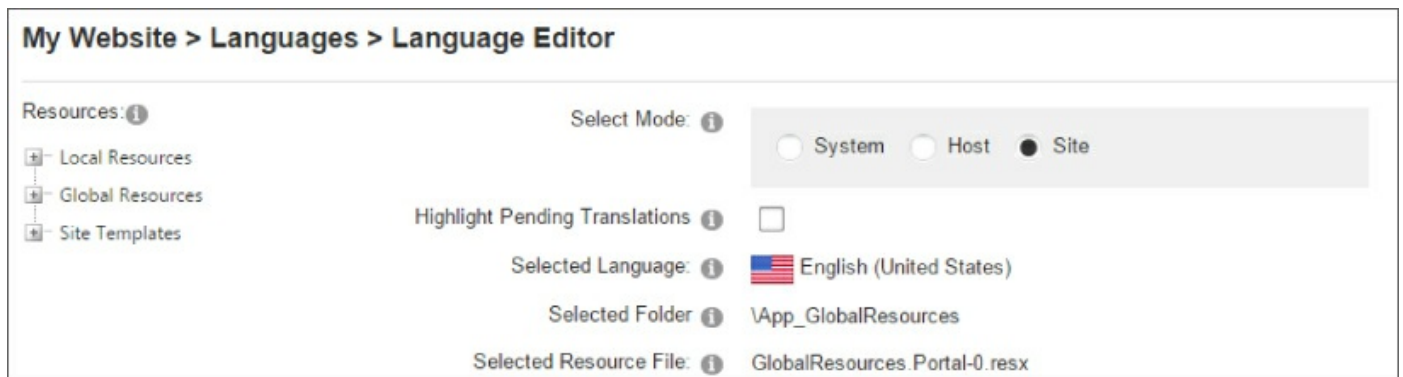


Figure 10.3

The language fallback is customizable by the administrator: each language that is added to DNN can be configured with a custom fallback language (the standard fallback language is the System Language, en-US).

[Figure 10.4](#) shows the Edit Language option, where host users can choose a fallback language. In this case, German (Germany) is chosen as a fallback for German (Austria).



Figure 10.4

Let's look at two examples of the fallback scheme. First a simple one. Assume

a single-language environment and one site, where a resource key is requested for a module control. DNN starts at the most specific file and ends up with the least specific one. If a key is not found in any of the files, it will be reported missing (using a log entry), and an empty value will be used. This is the order at which files will be tried:

- **Control specific:**

```
/DesktopModules/ModuleName/ControlName.Portal-1.resx
```

```
/DesktopModules/ModuleName/ControlName.Host.resx
```

```
/DesktopModules/ModuleName/ControlName.resx
```

- **Module shared resources:**

```
/DesktopModules/ModuleName/SharedResources.Portal-1.resx
```

```
/DesktopModules/ModuleName/SharedResources.Host.resx
```

```
/DesktopModules/ModuleName/SharedResources.resx
```

- **Application shared resources:**

```
/App_GlobalResources/SharedResources.Portal-1.resx
```

```
/App_GlobalResources/SharedResources.Host.resx
```

```
/App_GlobalResources/SharedResources.resx
```

The next example assumes Austrian (de-AT) as the requested language, having German (de-DE) defined as the fallback language. The fallback language of German is en-US (the System Language). The total number of fallback steps become much more complex now:

- **Control specific**

Requested language

- /DesktopModules/ModuleName/ControlName.de-AT.Portal-1.resx

- /DesktopModules/ModuleName/ControlName.de-AT.Host.resx

- /DesktopModules/ModuleName/ControlName.de-AT.resx

Fallback language

- /DesktopModules/ModuleName/ControlName.de-DE.Portal-1.resx

- /DesktopModules/ModuleName/ControlName.de-DE.Host.resx

- `/DesktopModules/ModuleName/ControlName.de-DE.resx`

System language

- `/DesktopModules/ModuleName/ControlName.Portal-1.resx`
- `/DesktopModules/ModuleName/ControlName.Host.resx`
- `/DesktopModules/ModuleName/ControlName.resx`

• Module shared resources

Requested language

- `/DesktopModules/ModuleName/SharedResources.de-AT.Portal-1.resx`
- `/DesktopModules/ModuleName/SharedResources.de-AT.Host.resx`
- `/DesktopModules/ModuleName/SharedResources.de-AT.resx`

Fallback language

- `/DesktopModules/ModuleName/SharedResources.de-DE.Portal-1.resx`
- `/DesktopModules/ModuleName/SharedResources.de-DE.Host.resx`
- `/DesktopModules/ModuleName/SharedResources.de-DE.resx`

System language

- `/DesktopModules/ModuleName/SharedResources.Portal-1.resx`
- `/DesktopModules/ModuleName/SharedResources.Host.resx`
- `/DesktopModules/ModuleName/SharedResources.resx`

• Application shared resources

Requested language

- `/App_GlobalResources/SharedResources.de-AT.Portal-1.resx`
- `/App_GlobalResources/SharedResources.de-AT.Host.resx`
- `/App_GlobalResources/SharedResources.de-AT.resx`

Fallback language

- `/App_GlobalResources/SharedResources.de-DE.Portal-1.resx`
- `/App_GlobalResources/SharedResources.de-DE.Host.resx`

- /App_GlobalResources/SharedResources.de-DE.resx

System language

- /App_GlobalResources/SharedResources.Portal-1.resx
- /App_GlobalResources/SharedResources.Host.resx
- /App_GlobalResources/SharedResources.resx

As you see, reading from resource files in a multi-lingual environment is quite complex. Even so, the performance of all this is still good, as DNN uses a cached dictionary that is used to keep track of whether or not a resource file actually exists. So even though the fallback scheme is very complex, a file gets tried only once. And in real-life situations it is quite rare for a site to have so many actual fallback files defined. Also, the contents of each resource file will be cached, so per file, there will be only one disk operation.

The Resource File Editor

DNN comes with its own resource file editor, which automatically keeps track of the System – Host – Site fallback scheme. It is a useful tool for site admins and translators alike to update resources or to create new translations of existing resources.

[Figure 10.5](#) shows how to access the editor from the Language Management module (located at Admin → Languages).

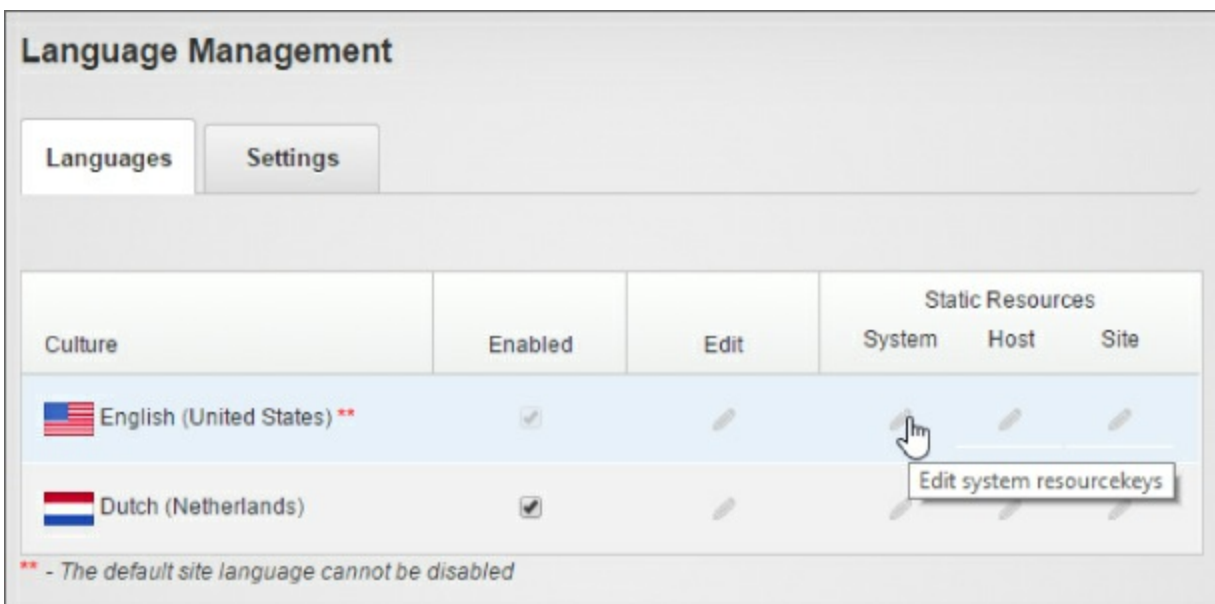


Figure 10.5

Clicking the edit pencil for the System, Host, or Site option will open a new screen that gives a full view of all resource files for the chosen language (see [Figure 10.6](#)).

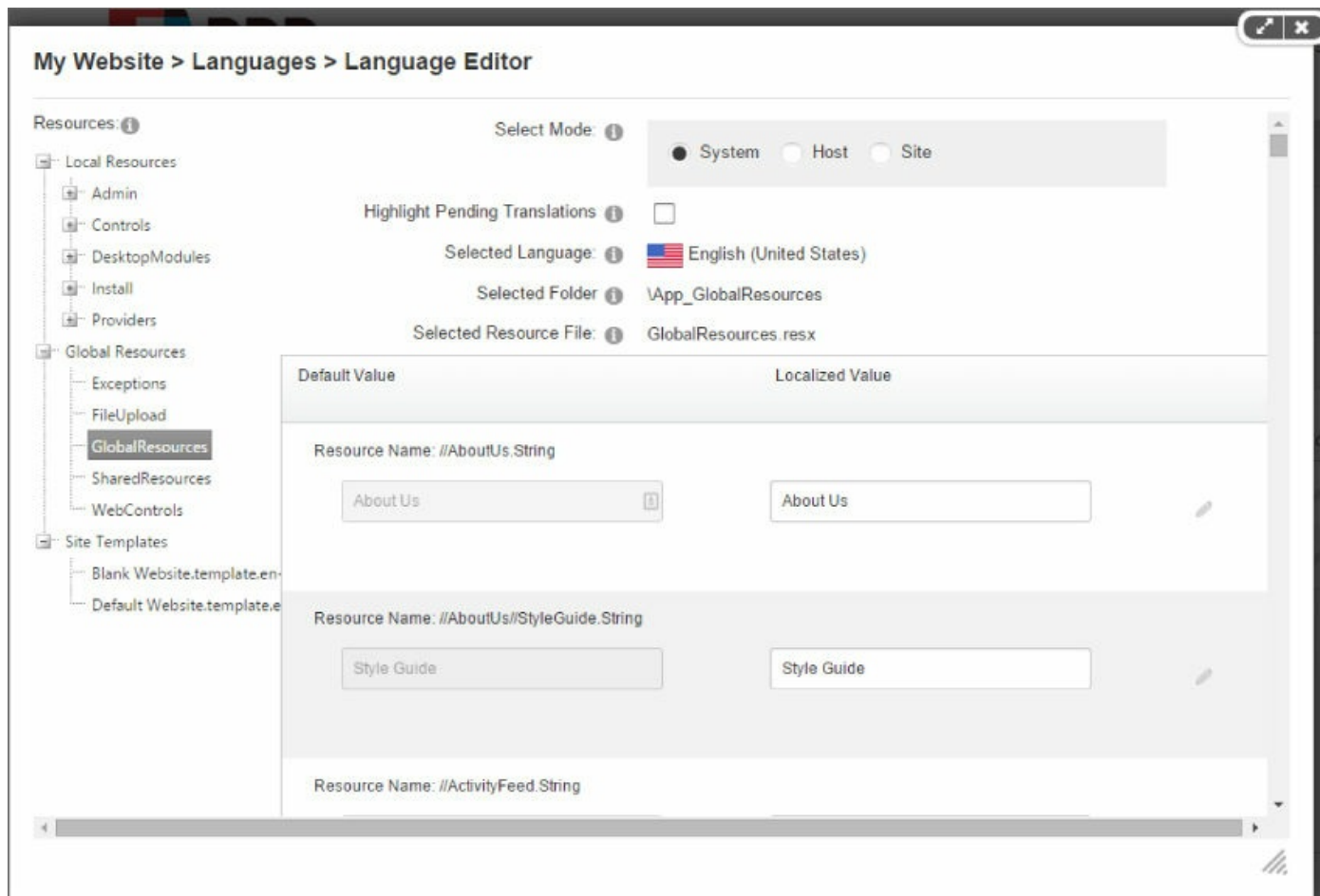


Figure 10.6

Missing Resource Keys

Keeping track of resource keys can be a daunting task, especially since there is no feedback in the UI about whether or not static text is properly localized. DNN offers different tools that give you more information about this.

The first tool is Log4Net, which is a logging tool. Whenever a request is made to the API to retrieve a string from a resource file, a warning is logged in the log file if the requested key was not found. In order to see these warnings, the Log4Net log level needs to be set to at least `WARN`. An example of such a warning is this:

```
2014-07-1 14:51:43,238 [DNNTTEST][Thread:8][WARN]
DotNetNuke.Services.Localization.
LocalizationProvider - Missing localization key.
```

```
key:ControlTitle_.Text
resFileRoot:/DesktopModules/DNNTest/DNNTestModule/App_LocalResources/\
threadCulture:en-US userlan:
```

This message contains all the information a developer needs to add the missing key to the resource file.

The second tool is something that is built into the framework and allows you to see localization information in the UI. The setting for this is called `ShowMissingKeys`, and it is located in the `appSettings` section of `web.config`. When set to `true`, you will see one of the following effects in the HTML output of the site, on every static text element:

- **No difference:** Text is not localizable.
- **[L]: prefixed to text:** Text is localizable, and value was read from resource file.
- **RESX: [KEY]:** Text is localizable; however, the key was not found in the resource file.

Sometimes functionality may be impacted by turning on `ShowMissingKeys`. You can counteract that in your code by using an API method that will ignore the `ShowMissingKeys` setting. This API method is called `GetStringURL`. You'll learn more about that in the next section.

The API

The main class that is used for localization-related tasks is `DotNetNuke.Services.Localization.Localization`. The most common methods are listed in [Table 10.1](#).

Table 10.1 Localization Methods

Method	Description
<code>GetResourceFile</code>	Returns the path and filename of the resource file for a specified control.
<code>GetString</code>	Returns the localized string based on the specified resource key.
<code>GetStringUrl</code>	Returns the localized string based on the specified resource key but ignores the <code>ShowMissingKeys</code> setting.
<code>GetSafeJSString</code>	Returns the localized, and escaped, string based on the specified resource key, specifically to be used in JavaScript.
<code>GetSystemMessage</code>	Localizes a string and replaces system tokens with personalized strings.
<code>LocalizeDataGrid</code>	Localizes the headers in a <code>DataGrid</code> control.
<code>LocalizeGridView</code>	Localizes the headers in a <code>GridView</code> control.
<code>LocalizeDetailsView</code>	Localizes the headers in a <code>DetailsView</code> control.
<code>SetThreadCultures</code>	Sets the culture code for the current thread. This needs to be done for pages that do not derive from <code>PageBase</code> .

The GetString Method

Of the methods in [Table 10.1](#), the most widely used is `GetString`. It performs a localized lookup based on the resource key passed into it. `GetString` has eight overloaded methods:

```
public static string GetString(string key)
public static string GetString(string key, Control ctrl)
public static string GetString(string key, PortalSettings
portalSettings)
public static string GetString(string key, string
resourceFileRoot)
public static string GetString(string key, string
```

```

resourceFileRoot,
    bool disableShowMissingKeys)
    public static string GetString(string key, string
resourceFileRoot,
    string language)
    public static string GetString(string key, string
resourceFileRoot,
    PortalSettings portalSettings, string language)
    public static string GetString(string key, string
resourceFileRoot,
    PortalSettings portalSettings,
    string language, bool disableShowMissingKeys)

```

The overloads work from generic to specific. If no `resourceFileRoot` is passed in, DNN will only try to get the requested key from the application-level shared resources file (`/App_GlobalResources/SharedResources.resx`). Normally you should not have to use an overload that requires a `PortalSettings` instance; the other overloads always use the current `PortalSettings` object. This does however mean that it is not safe to use any of these methods when no site context is available (for instance, from a scheduled task).

[Table 10.2](#) lists all possible parameters and their descriptions.

Table 10.2 GetString Parameters

Parameter	Type	Description
<code>Ctrl</code>	Control	This is the object representing an ASCX user control and is passed as <code>this</code> in C# or <code>Me</code> in VB.NET.
<code>disableShowMissingKeys</code>	Boolean	Allows for overriding the setting to display missing localization keys.
<code>Key</code>	String	The key for which a resource value is requested.
<code>Language</code>	String	The name of the language used to look up the string.
<code>portalSettings</code>	PortalSettings	The <code>PortalSettings</code> object for the current context. It is used to determine the default locale used for anonymous users.
<code>resourceFileRoot</code>	String	The value of a module's

		LocalResourceFile property. It is used to derive the resource file to be used for the translation.
--	--	--

The GetMessage Method

The `GetSystemMessage` method is used throughout the core code to produce localized and personalized strings. For example, it is frequently used to send email to users who register on a site. The user registration page calls `GetSystemMessage` to retrieve the language-appropriate welcome content of the email and then replaces tokens for site- and user-specific personalization. `GetSystemMessage` gives developers a tool for localization and token replacement personalization with just one method call.

`GetSystemMessage` has 10 overloaded methods:

```
public static string GetMessage(PortalSettings portalSettings,
                               string messageName)
public static string GetMessage(PortalSettings portalSettings,
                               string messageName, UserInfo userInfo)
public static string GetMessage(string strLanguage,
                               PortalSettings
                               portalSettings, string messageName, UserInfo
                               userInfo)
public static string GetMessage(PortalSettings portalSettings,
                               string messageName, string resourceFile)
public static string GetMessage(PortalSettings portalSettings,
                               string messageName, UserInfo userInfo,
                               string resourceFile)
public static string GetMessage(PortalSettings portalSettings,
                               string messageName, string resourceFile,
                               ArrayList custom)
public static string GetMessage(PortalSettings portalSettings,
                               string messageName, UserInfo userInfo,
                               string resourceFile, ArrayList custom)
public static string GetMessage(string strLanguage,
                               PortalSettings
                               portalSettings, string messageName, UserInfo
                               userInfo,
                               string resourceFile, ArrayList custom)
public static string GetMessage(string strLanguage,
                               PortalSettings
                               portalSettings, string messageName, UserInfo
                               userInfo,
                               string resourceFile, ArrayList custom, string
                               customCaption, int accessingUserID)
public static string GetMessage(string strLanguage,
                               PortalSettings
```

```

userInfo,
portalSettings, string messageName, UserInfo
IDictionary
string resourceFile, ArrayList customArray,
customDictionary, string customCaption,
int accessingUserID)

```

[Table 10.3](#) lists all possible parameters and their descriptions.

Table 10.3 GetSystemMessage Parameters

Parameter	Type	Description
portalSettings	PortalSettings	The <code>PortalSettings</code> object for the current context. It is used to derive any personalized content in the localized system message string.
messageName	String	The resource key used to get the localized system message from the resource file. Because no user information is included in this overload, “User:” <code>MessageName</code> types are not supported.
userInfo	UserInfo	The <code>UserInfo</code> object for the current context. It is used to derive any personalized content in the localized system message string.
strLanguage	String	The name of the language used to look up the string.
resourceFile	String	The resource file that the localized system message is stored in.
custom	ArrayList	A collection of strings that can be used for personalizing the system message.
customCaption	String	Property to override the default text used to access custom properties.
accessingUserID	Integer	The user ID of the user accessing the system message—used to restrict unauthorized access to profile properties during the token replacement process.
customDictionary	IDictionary	A custom dictionary that can be used

instead of the `custom` parameter.

The method can be used to parse custom message templates against both system tokens and custom tokens. Internally, a call is made to the DNN `TokenReplace` engine. When you use `GetSystemMessage`, you can specify several system tokens in the localized string. The tokens are used as keys to render property values from the `UserInfo` or `PortalSettings` objects. Here's an example:

```
body = Localize.GetSystemMessage(locale, settings, body, user,
    Localize.GlobalResourceFile, custom, "",
    settings.AdministratorId);
```

where `Localize` is an alias to

`DotNetNuke.Services.Localization.Localization`. This code calls the `GetSystemMessage` method to generically localize and personalize the body of email messages that can be sent to users for a variety of reasons. The code is from the `SendMail` method of the `DotNetNuke.Services.Mail` class.

The method is used to handle different types of messages. Let's look at one example, for emails sent to verify user registrations. The different parameters of the call to `GetSystemMessage` are initialized with these values:

```
body = "EMAIL_USER_REGISTRATION_VERIFIED_BODY";
var propertyNotFound = false;
if (HttpContext.Current != null)
{
    var server = HttpContext.Current.Server;
    custom = new ArrayList
        {
            server.HtmlEncode(server.UrlEncode(user.Username)),
            server.UrlEncode(user.GetProperty("verificationcode",
                String.Empty, null, user, Scope.SystemMessages,
                ref propertyNotFound))
        };
}
```

`EMAIL_USER_REGISTRATION_VERIFIED_BODY` is the resource key to look up the system message in the resource file. The resource key and associated translation from the global resource file

`/App_GlobalResources/GlobalResources.resx` are shown in the following code:

```
<data name="EMAIL_USER_REGISTRATION_VERIFIED_BODY.Text"
```

```
xml:space="preserve">
  <value>Dear [User:DisplayName],
We are pleased to advise that you have been added as a Registered
User to
[Portal:PortalName]. Please read the following information carefully
and be
sure to save this message in a safe location for future reference.
Portal Website Address: [Portal:URL]
Username: [User:UserName]
You can use the following link to complete your verified
registration:
Error! Hyperlink reference not valid.;
verificationcode=[Custom:1]
Thank you, we appreciate your support...
[Portal:PortalName]
  </value>
</data>
```

`GetSystemMessage` first localizes the string in the `<value>` XML node. Then it iterates through the system tokens (enclosed in brackets), replacing the tokens with the appropriate property values. For example, in the following code example, you can see the token `[User:DisplayName]`. This will be replaced with the `DisplayName` property value of the `User` object. In this case, the `User` object is the `user` object passed into the `GetSystemMessage` method. There is also an array being passed into the method, the contents of which are accessed through these tokens: `[Custom:0]` and `[Custom:1]`.

The following code shows that the system message has been personalized and localized with the `en-US` locale:

```
Dear John Doe,

We are pleased to advise that you have been added as a Registered
User to
My Website. Please read the following information carefully and be
sure
to save this message in a safe location for future reference.

Portal Website Address: test.dnndev.me/en-us
Username: JohnDoe

You can use the following link to complete your verified
registration:

http:// test.dnndev.me/en-us/default.aspx?ctl=Login&username=JohnDoe&
verificationcode=IeOYlcFhJGg_

Thank you, we appreciate your support...
```

Token Replacement Engine

DNN features a token replacement engine that offers functionality to personalize localized templates. This is used in several areas of the DNN Platform, but it can also be leveraged in your modules. The engine goes beyond simple token replacement to add formatting and analysis that allows administrators to have more control over the display of token replaced values. A few examples of module that use this feature are the core Newsletters and Text/HTML modules, Forms & Lists, and Announcements. Another module that leverages token replacement is the Blog module. The Blog module, however, builds on token replacement to make it better suited for template handling.

To use tokens in system messages and the modules that implement the service, add the token to your text in the `[Object:Property]` format. For example, to display a user's display name in an email message, you add `[User:DisplayName]`. Formatting and conditional replacement can be done using one of the following syntax options:

- `[Object:Property]`
- `[Object:Property|Format]`
- `[Object:Property|Format|IfEmptyReplacement]`

Consider the following salutation from an email sent through the Newsletters module:

```
[Profile:Region|Dear {0} Resident|Dear Valued Member]
```

Users with a value stored in the `Region` property of their `Profile` framework object would be saluted as *Dear Florida Resident*, whereas users who have no value in the `Region` property would be saluted as *Dear Valued Member*.

Properties are formatted with the ability to use the same methods found in the .NET Framework for `Date`, `String`, and `Integer` types. This allows for arguments to be passed as formatting conditions. Consider the following text that can be used to show the current site's logo with the time displayed directly beneath:

```
 [Date:Now|MM-dd-yyy h:mm:ss]
```

[Table 10.4](#) lists all the objects that are available for token replacement. Appendix B includes lists all accessible properties for these objects.

Table 10.4 Objects Available for Token Replacement

Object	Class/Data Source	Returns
Host	System.Collection.Hashtable	Secure HostSettings
Portal	DotNetNuke.Entities.Portals.PortalSettings	Current PortalSettings
Tab	DotNetNuke.Entities.Tabs.TabInfo	Current TabInfo
Module	DotNetNuke.Entities.Modules.ModuleInfo	Current Module
Culture	System.Globalization.CultureInfo	Current Culture
User	DotNetNuke.Entities.Users.UserInfo	Current User
Profile	DotNetNuke.Entities.Profile	Current User.Profile
Membership	DotNetNuke.Entities.Users.Membership	Current User.Membership
Role	DotNetNuke.Security.Roles.RoleInfo	Current Role
Date, DateTime, Time	System.DateTime	Current DateTime
Ticks	System.Int64 (Long)	Current DateTime in ticks
Row, Field	System.Data.DataRow	Not applicable
Custom ¹	System.Collections.ArrayList	Not applicable
Custom ¹	Any System.Collection.IDictionary	Not applicable
User Defined Token	Any Object by Reflection	Not applicable

* Custom can be replaced with custom text, passed as a separate parameter.

Localizing Modules

The current version of DNN supports localizing any static text in the user interface of a module. When you are distributing your modules publicly, it is important to prepare the UI of your module for localization so that it can be used for non-English-speaking sites. Imagine the frustration of your users if all the content for their site is written in Spanish, yet all of the static text in your module is written in English.

Though the localization API is very powerful, it would add a lot of work for module developers if it were the only mechanism for localization. To simplify the job for developers, DNN includes a localization framework that applies localization using declarative markup in the ASCX or ASPX files. This approach provides a couple of benefits:

- It simplifies the programming model because the developer adds a single attribute/value pair to the server control markup, and the framework handles calling the appropriate localization APIs.
- It allows localization to be applied or changed without recompiling the application.

So look at what it takes to provide localization for your module.

After all of the static strings have been identified, you must determine the best approach for localizing each individual string. Each string should be categorized into one of four cases depending on how the string is used within a module:

- Text placed directly into the HTML in the ASCX files
- Text declaratively set in a server control in the ASCX files
- Text modified or set in the source code for the module
- Text embedded in images

The following sections show you how to use the localization framework and localization API to correct each of these potential problem areas.

Case 1: Handling Static Strings in the ASCX File

The key to resolving the problem of text placed directly into the HTML in the ASCX files is to understand that localization is handled programmatically. Whether it is your code or framework code, you need to have the string in an

element that is easily accessed programmatically. This means that you must make sure that the code-behind file and the DNN framework are aware of the string's existence. This is actually quite easy to fix; just wrap the string in an HTML control. Essentially, this step transforms the problem into Case 2 and enables you to use a common approach for all strings in the ASCX file. [Table 10.5](#) shows an example of applying this step to a simple string.

Table 10.5 Wrapping a String with a Web Control

Before	After
<pre><div class="SubHead"> Title: </div></pre>	<pre><div class="SubHead"> <asp:Label id="lblTitle"runat="server">- Title: </asp:Label> </div></pre>

At this point, your strings are ready to be localized.

Case 2: Handling Static Text in Server Controls

After all of your static strings are encapsulated in server controls, it is easy to tell the localization framework how to localize your strings. To localize a control, add a `resourcekey` attribute with a value that tells the framework which string resource to use for this control. [Table 10.6](#) takes the previous example and makes this additional change.

Table 10.6 Adding a Resource Key

Before	After
<pre><div class="SubHead"> <asp:Label id="lblTitle" runat="server" Title:- </asp:Label></div></pre>	<pre><div class="SubHead"> <asp:Label id="lblTitle" runat="server" resourcekey="TestLabel"> Title:- </asp:Label></div></pre>

Now, the framework has all of the information it needs to find your string resources and localize this content.

As mentioned earlier in the chapter, your module should include the `App_LocalResources` directory with a resource file named after the user control that is being localized.

To verify that you have applied the localization settings correctly, ensure that the `web.config` file's `AppSettings` section includes the following line:

```
<add key="ShowMissingKeys" value="true"/>
```

If you have applied the `resourcekey` and set `ShowMissingKeys` correctly, you should see the image in [Figure 10.7](#), the localized string with a missing value.

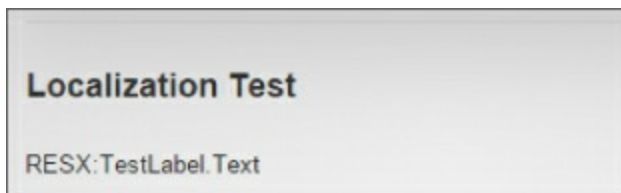


Figure 10.7

Notice that the text shows you that it is looking for `TestLabel.Text`. This makes it easy to see where you have localized a control and where you still need to create the localized version of the string.

A Label control shows how easy it can be to localize a web control. But each control is different and contains different attributes that might need to be localized. [Table 10.7](#) shows which attribute will be localized using the default behavior.

Table 10.7 Default Localized Attributes

Control	Type	Localized Attribute
System.Web.UI.WebControls	Label	Text
	Button	Text
	LinkButton	Text
	ImageButton	AlternateText
	Hyperlink	Text
	Image	AlternateText
	CheckBox	Text
	BaseValidator	ErrorMessage
	BulletedList	Items[i].Text
	CheckBoxList	Items[i].Text
	DropDownList	Items[i].Text
	ListBox	Items[i].Text
	RadioButtonList	Items[i].Text
System.Web.UI.HtmlControls	HtmlImage	Alt
	HtmlButton	Title

If the web control you are localizing is not listed in [Table 10.7](#), you need to localize a different attribute, or if you need to localize multiple attributes for the same control, you must use the localization techniques shown in Case 3.

Case 3: Handling Static Text Programmatically

You will find several cases where text cannot be localized using a declarative approach. This will require programmatically setting the text using the localization API. The API section outlines the most frequently used methods of the API.

Continuing with the same example, take a look at how to handle localizing the `ToolTip` attribute for the label you created. Notice that this attribute is not the default attribute for the Label control listed in [Table 10.7](#). To localize the attribute, add the following line of code to the code-behind:

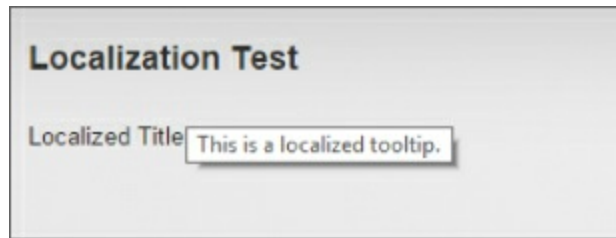
```
lblTitle.ToolTip = LocalizeString("TestLabel.ToolTip")
```

This will use the shortcut to `Localization.GetString`, available at the level of a module control. This will use the automatically generated filename for your module control, which is

`ControlPath/App_LocalResources/ControlName.ascx.resx`. If you are writing code that is not placed inside a DNN module control, then you will have to define your own file path for the resource file and use a version of `GetString` from the `Localization` class.

Because this is a new control, you are safe adding it to the `Page_Load` event. As you will see shortly, this is not always the appropriate spot for localizing strings. Now that you have the code, the only step left is to add the localized value to the resource file. This example uses `LocalResourceFile` because it is specific to the module. Also note that although the key is named `TestLabel.ToolTip`, you are free to use whatever key makes sense to you. Because you're using the API, you have much more control over how the keys are named.

If your key does not include a period (`.`), the framework will automatically add `.Text` to your key and use that as the key for looking up the localized value. Keep this in mind when creating your keys. Assuming that you have named your key appropriately, you should see something like the localized tooltip shown in [Figure 10.8](#) when you compile and navigate to a page with your module.



[Figure 10.8](#)

There are many instances where you might have embedded strings in your code that are changed depending on the application state. A good example of this is the `Login` skin object. This skin object changes between `Login` and `Logout` depending on the authentication state of the current user. To localize this control, just replace all the references to the static text with a call to one of the `GetString` methods described earlier.

Images are a special case. It is recommended that you never include text in your images because it complicates localization and can usually be avoided through the use of background images and CSS. If your design requires you to embed text in an image, you need to make a few changes to make it easier to localize the image. To embed text into images and switch them with CSS, override the styles from the base `module.css` stylesheet in a separate stylesheet and switch them dynamically using the following code in the `OnInit` event handler of your user control. You should use the `ClientResourceManager` to do so, to make sure that registration takes place in the head of the HTML output:

```
ClientResourceManager.RegisterStyleSheet(Page,  
    string.Format("{0}CSS/languagespecific-  
    {1}.css",  
    ControlPath, CultureInfo.CurrentUICulture.Name));
```

For the links module, this would render like this with the default U.S. English language:

```
<link href="#/DesktopModules/Links/CSS/languagespecific-en-US.css?  
cdv=1"  
media="all" type="text/css" rel="stylesheet"/>
```

No module will ever include localized resources for every language supported by DNN. The effort to maintain the resource files would greatly exceed the cost for all other development. Most module developers will include resource files for their native language and maybe one or two other languages depending on the language skills of the module development staff. This means that many users will be forced to create the resource files for their own

language. That's not usually a significant problem.

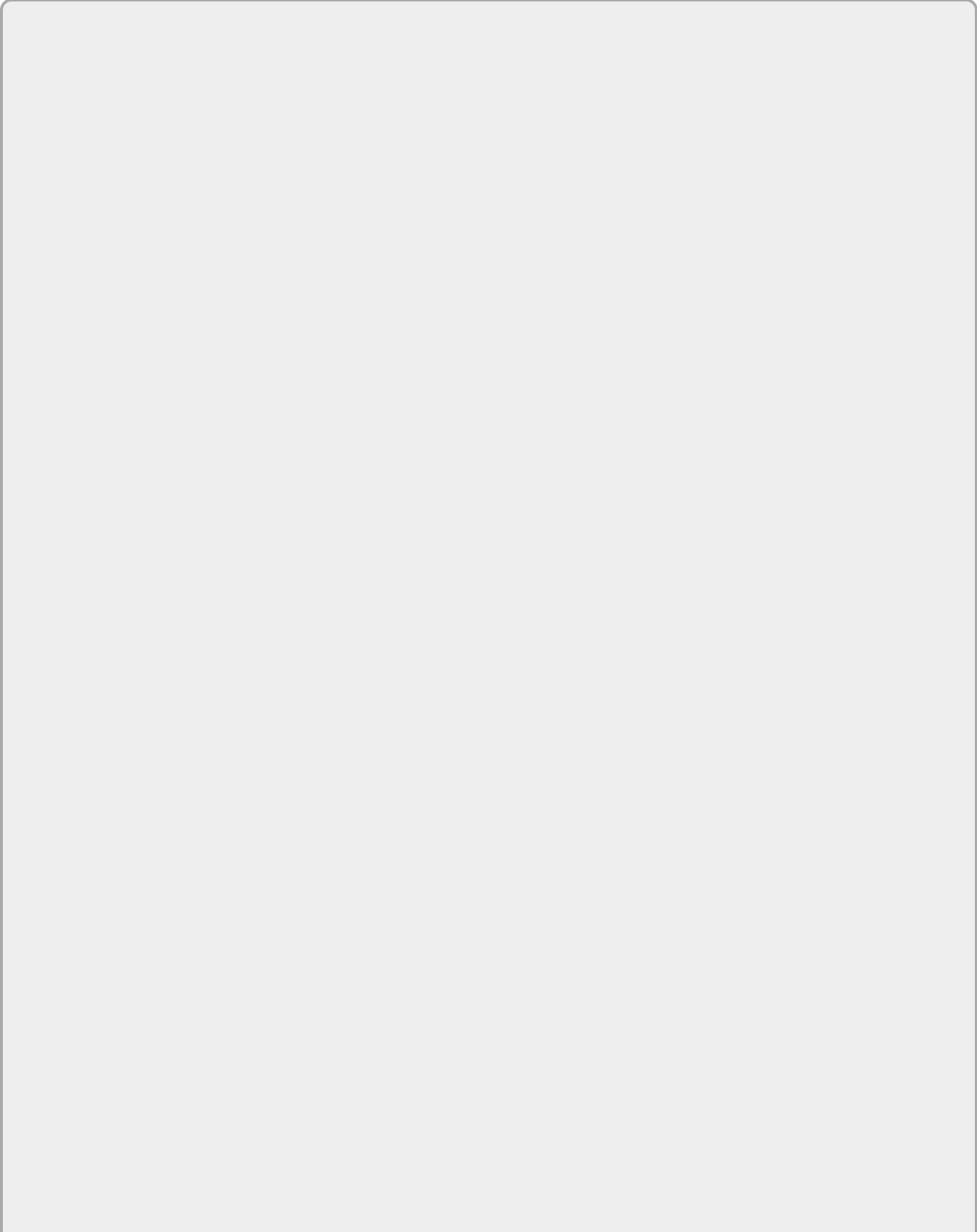
If an image file contains embedded text, though, the user is forced to re-create the image with a localized version of the text. To ease the burden for the user, include the base image without text so that the user can easily create the text label in his language. Keep in mind that different languages have different space requirements. Just because a word or phrase is short in one language does not mean that it will be equally short in another language. That's one of the reasons for avoiding embedded text.

When your users have the image file with the localized text, they can use the standard `GetString` methods to set the `ImageUrl` attribute to the appropriate image filename, depending on the language they select.

Summary

In this chapter, you learned how resource strings are stored and used for the core framework and for each module. You reviewed how locale and language detection work in the DNN framework to provide a base for modules to build upon to provide localized interfaces.

Chapter 11
Search



What's in this Chapter?

- Learning about Lucene
- Discovering new features in search
- Understanding search entities and APIs
- Integrating with modules
- Writing a new crawler

Search was built from the ground up in DNN 7.1. The main objective was to provide Google-type search capabilities in the DNN platform and the extensions that were built on top of that. Some of the key features of search were speed, relevance, and security.

History

The DNN Platform has had search functionality since the early versions. Search functionality was based on the `ISearchable` interface, where a scheduled task was present to periodically probe modules implementing this interface. The interface allowed the platform to extract information from modules and store it in a common store. Search storage happened to use SQL Server, which provided a good centralized location for storing and querying content. Although SQL Server provided decent search capabilities, it had many problems such as speed, accuracy, and result highlighting.

When the commercial editions were initially launched around DNN 5.x, they included the File Crawler and URL Crawler. These were also implemented as scheduled tasks. The File Crawler allowed indexing of Office and PDF documents. The URL Crawler was more like a typical web crawler and traversed from one link to another on the site, thereby parsing and indexing the content.

Objectives of the New Search Functionality

Supporting two different search architectures across editions was very taxing. Differentiation makes a lot of sense at the feature level, but not at the architecture level. There was also need for a more efficient search API for modules. For example, the concept of *deltas* was introduced, where modules can recognize changes in content since the last indexing run, as opposed to indexing the entire content all of the time. Several of the key objectives of the new search functionality included:

- Handling diverse content (CMS content, social content, localized content, and third-party modules)
- Providing a consistent user experience across various solutions
- Providing simplicity for module developers
- Using a uniform architecture with differentiation based on feature, not architecture

Apache Lucene

Apache Lucene has been the de facto open source search engine for many years. It is written in Java and therefore can be run on a variety of platforms, including Windows. More information about Lucene can be obtained at <http://lucene.apache.org/>. [Figure 11.1](#) shows Lucene's logo. The DNN platform does not use the Java version of Lucene.



[Figure 11.1](#)

Lucene.Net

Lucene.Net is a line-by-line C# port of the original Java-based Lucene. It is open source as well and is part of the Apache Foundation. Similar to the original Lucene, the .Net version is also a very efficient searching library. More information about Lucene.Net can be obtained at <http://lucenenet.apache.org/>. The DNN platform uses Lucene.Net and ships the necessary DLLs in its installation package. [Figure 11.2](#) shows Lucene.Net's logo.



[Figure 11.2](#)

Lucene: A Document Store

Internally, Lucene is a *document store* consisting of a collection of documents. Each document consists of a collection of *fields*. In comparison with a typical relational database, a document can be compared to a table and a field to a column. A document does not have a name per se; however, a field has a name, type, and value. Lucene, in fact, supports a flexible schema, meaning there is no requirement for two different documents to have the same set of fields.

Although Lucene is not marketed as NoSQL, it is indeed a NoSQL database on the inside. People at Lucene prefer that Lucene be known for its search capabilities and not necessarily as a document store or a NoSQL database.

Many organizations (for example, Facebook, LinkedIn, and Netflix) utilize

Lucene or a variant of it to provide search capabilities in their applications. Lucene is so powerful as a document store that RavenDB (<http://ravendb.net/>), a NoSQL database for .Net, uses Lucene.Net as its backend.

Lucene versus Lucene.Net

One thing that's very important to note is that Lucene.Net and Lucene are not the same. The concepts behind them are the same; in fact, the .Net version is a direct port of the Java version.

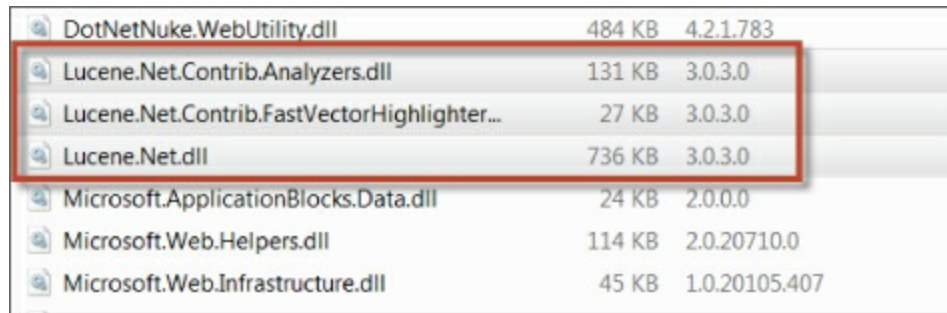
Besides the fact that the programming languages are different, the release schedule is vastly different as well. The Java version ships more frequently than the .Net version. The latest version of Lucene.Net, 3.0.3, was released in October 2010, whereas the latest version of Lucene, 4.9.0, was released in June 2014. It is also to be noted that the Java version of 3.0.3 was released in March 2011, which means that the .Net version was behind by close to 18 months. According to Lucene.Net's wiki, the next two releases are going to be 3.6 and 4.0. No dates have been specified.

Although the Lucene.Net 3.0.3 version may look older compared to Lucene, 3.0.3 is still a very good release of Lucene.Net. In fact, there is a very popular book written on version 3.0: *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. The book has about 500 pages and provides discussions on many aspects of Lucene. This book was extensively referenced in building search functionality in DNN 7.1.0.

Lucene.Net Contrib

Besides Lucene.Net, there are a few helper DLLs (Contrib) available to perform specialized functionality outside of the core Lucene.Net. Both Lucene.Net and the Contrib DLLs can be downloaded from NuGet, but they are also included in the DNN platform package. Similar to Lucene.Net, the Contrib DLLs are also part of the Apache Software Foundation.

DNN currently ships version 3.0.3 of Lucene.Net.dll as well as Lucene.Net.Contrib.Analyzers.dll and Lucene.Net.Contrib.FastVectorHighlighter.dll. As shown in [Figure 11.3](#), the three DLLs are relatively small in size, yet they are extremely powerful.



A screenshot of a file list showing several DLL files. A red rectangular box highlights the following three entries:

DotNetNuke.WebUtility.dll	484 KB	4.2.1.783
Lucene.Net.Contrib.Analyzers.dll	131 KB	3.0.3.0
Lucene.Net.Contrib.FastVectorHighlighter...	27 KB	3.0.3.0
Lucene.Net.dll	736 KB	3.0.3.0
Microsoft.ApplicationBlocks.Data.dll	24 KB	2.0.0.0
Microsoft.Web.Helpers.dll	114 KB	2.0.20710.0
Microsoft.Web.Infrastructure.dll	45 KB	1.0.20105.407

Figure 11.3

Search Architecture

As shown in [Figure 11.4](#), Lucene is located at the very bottom of the architecture stack of search. DNN developers usually don't need to know much about this layer as the layer above it abstracts Lucene technicalities away.

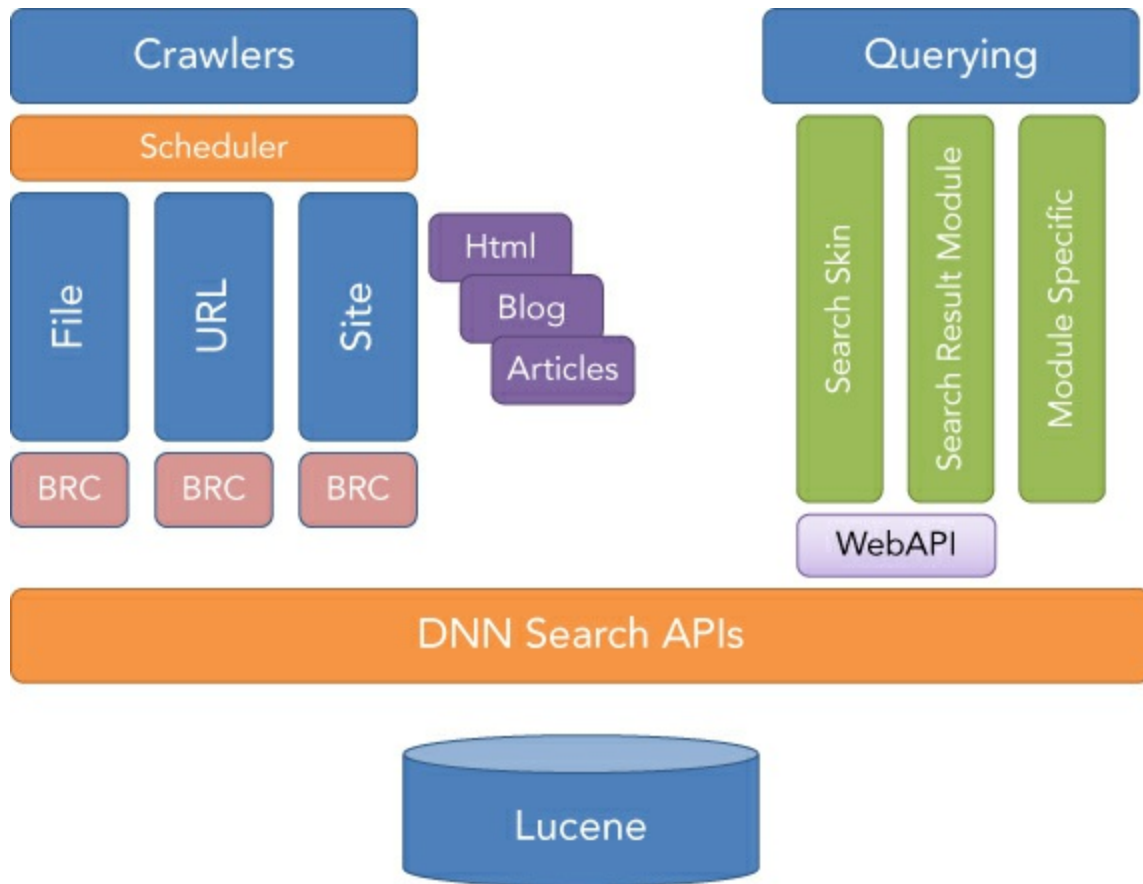


Figure 11.4

The DNN search APIs layer provides a way to store, delete, and find content from the Lucene index. The APIs work as a shim between Lucene and DNN. The APIs and their associated entities are DNN-friendly and are very easy to use.

Content gets added to the index through crawlers, which in turn are executed through the scheduler. The DNN Platform comes bundled with Site Crawler; the Evoq products have additional File and URL Crawlers. Site Crawler is responsible for calling into modules to get their content and store it in the search index. Each of the crawlers implements `BaseResultController` (BRC), which essentially is an abstract class containing two abstract methods

—`HasViewPermissions` and `GetDocUrl`.

Site visitors perform searches on the site through either the search skin object or the Search Results module. Both of them in turn call into search-specific Web APIs available in the platform to retrieve search results. These Web APIs can be called from outside of DNN as well. Many of the modules in the Evoq products implement their own querying user interface and call into search APIs directly.

Platform Features

The majority of the search features such as the use of Lucene.Net, near real-time querying, the ability to ignore words, synonyms, and re-indexing were incorporated into the base 7.1 platform itself. On top of the platform, the 7.1 Evoq modules took advantage of these functionalities and built additional module-specific search capabilities. The older crawlers in the commercial products were enhanced to take advantage of the new search functionality as well.

Site Crawler


Site Crawler is the scheduled task responsible for indexing the items presented in [Table 11.1](#). The indexed content is added to the Lucene data store. By default, the task runs every minute to ensure new content is indexed with very little latency. [Figure 11.5](#) shows the task listed in Host  Schedule. Further, [Figure 11.6](#) indicates the output once the task is completed. It shows the count of the items processed, which are explained in [Table 11.1](#). Site Crawler actually replaces the old Search Engine Scheduler task that was present in the platform prior to 7.1.

Table 11.1 Site Crawler's Tasks

Crawler	Task
Module's content	Indexes content from the module that implements either the old <code>ISearchable</code> or the new <code>ModuleSearchBase</code> . Also indexes content inside HTML tag attributes such as <code>ALT</code> and <code>TITLE</code> to search images, videos, and links.
Module's metadata	Indexes metadata for all the modules defined across pages on Sites and Host. Metadata includes module name, culture, header, footer, and taxonomy tags.
Tab's metadata	Indexes metadata for all the pages defined across Sites and Host. Metadata includes page name, title, description, headers, keywords, and taxonomy tags.
User's content	Indexes profile properties of all users including super users. This is introduced in version 7.2.
Journal's content	Indexes the journal. This is introduced in version 7.3.

Host > Schedule

Schedule

	Name	Enabled	Frequency	Retry Time Lapse	Next Start	Log
	Purge Users Online	<input type="checkbox"/>	Every 1 Minute	Every 5 Minutes		
	Purge Site Log	<input type="checkbox"/>	Every 1 Day	Every 2 Hours		
	Purge Schedule History	<input checked="" type="checkbox"/>	Every 1 Day	Every 2 Days	1/13/2015 3:50:23 PM	
	Purge Log Buffer	<input type="checkbox"/>	Every 1 Minute	Every 5 Minutes		
	Send Log Notifications	<input type="checkbox"/>	Every 5 Minutes	Every 10 Minutes		
	Search: Site Crawler	<input checked="" type="checkbox"/>	Every 1 Minute	Every 5 Minutes	1/12/2015 3:51:24 PM	
	Purge Cache	<input type="checkbox"/>	Every 2 Hours	Every 30 Minutes		
	Purge Module Cache	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds	1/12/2015 3:51:25 PM	
	Messaging Dispatch	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds	1/12/2015 3:51:26 PM	
	Purge Client Dependency Files	<input type="checkbox"/>	Every 1 Day	Every 6 Hours		

Figure 11.5

My Website > Schedule > Schedule History

You are using a trial version of Evoq Content Enterprise. You currently have 30

Description

Search: Site Crawler Starting. Content change start time 1/12/2015 3:52 PM

Tabs Indexed: 0
 Modules (Metadata) Indexed: 0
 Modules (Content) Indexed: 0
 Users Indexed: 0
 Deleted Objects: 0
 Total Items Indexed: 0
 Indexing Successful

Figure 11.6

Near Real-Time Searching

Lucene's near real-time functionality allows searching of documents within milliseconds of the documents getting added to the index. This feature is part of Lucene itself. There are two reasons why the prefix “near” is appropriate. First, Lucene is not truly real-time, and second, the crawling is done through a scheduled task, which has inherent delays. However, there are unsupported APIs in the DNN Platform to remove dependency of the scheduler and update

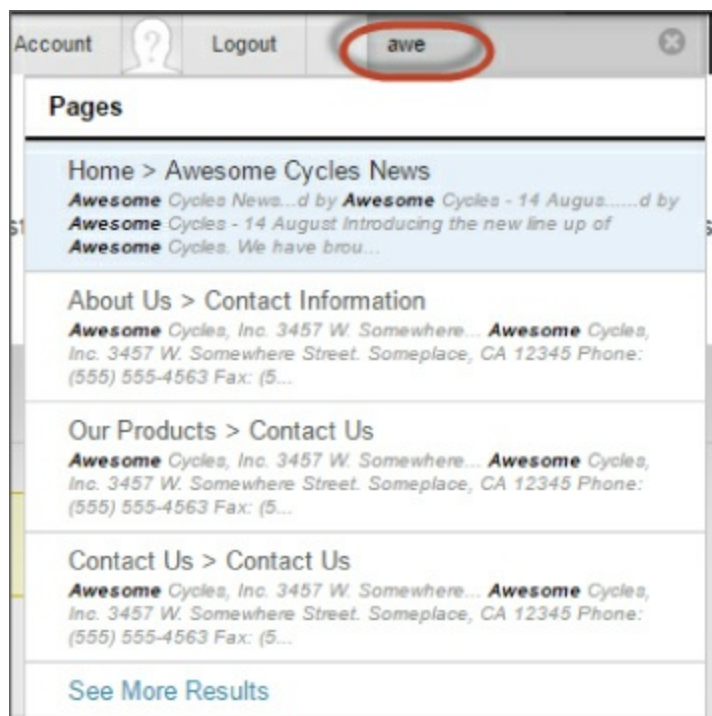
the index virtually in real time. These unsupported APIs have some limitations in a web farm. The initial indexing or re-indexing of a site can take longer; however, after that, it should be very quick as only new or changed content will be added to the Lucene store in the subsequent runs.

Speed

The new search functionality is extremely fast as it does not rely on SQL or out-of-process calls. Although Lucene does not market itself as NoSQL, it indeed is a NoSQL database. In fact, RavenDB, a document database, uses Lucene.Net as its backend. Because Lucene.Net ships as a DLL and runs as part of the DNN application, the calls for search are all in-process, which significantly increases performance as compared with the previous SQL implementation.

Auto-Preview

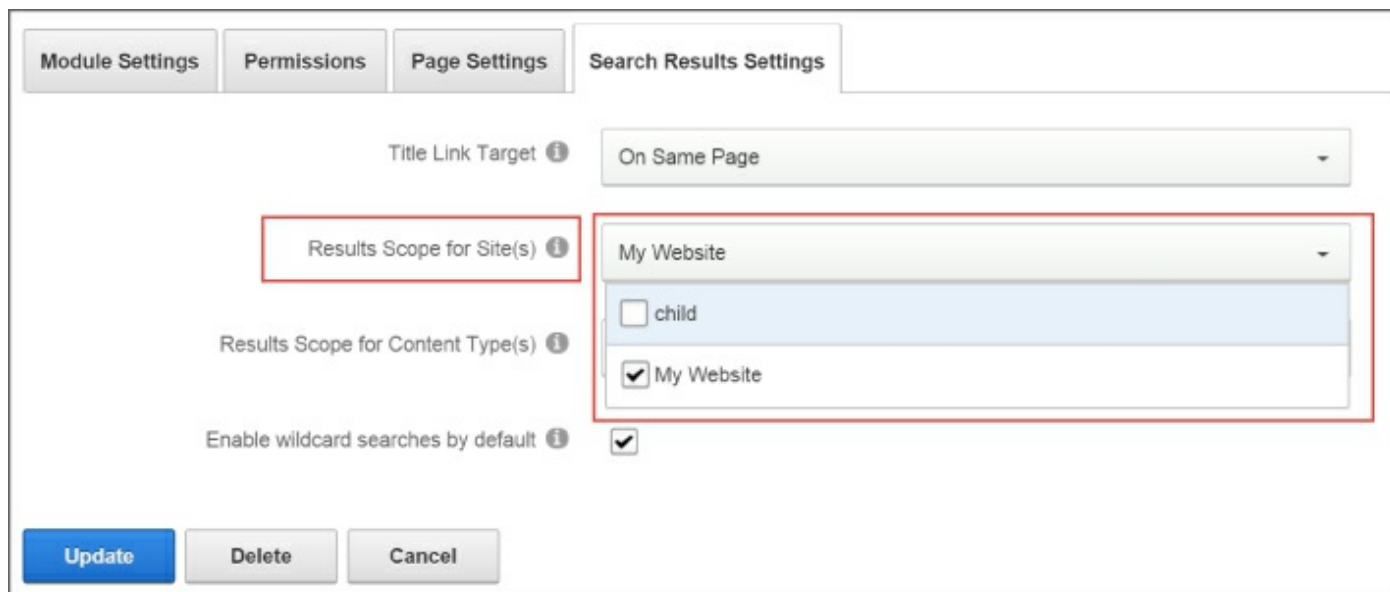
Both the search skin object and the Search Results module show search results as soon as the user starts to type search queries. Searching can be performed using partial words; the user doesn't need to type the entire word or phrase to get results. The main idea is to let users type a query, have a quick look at the results via a preview, and tweak the query as needed until they find the content they are looking for. [Figure 11.7](#) shows a user typing the partial word “awe” and the previews showing results.



[Figure 11.7](#)

Site-Scoped Search

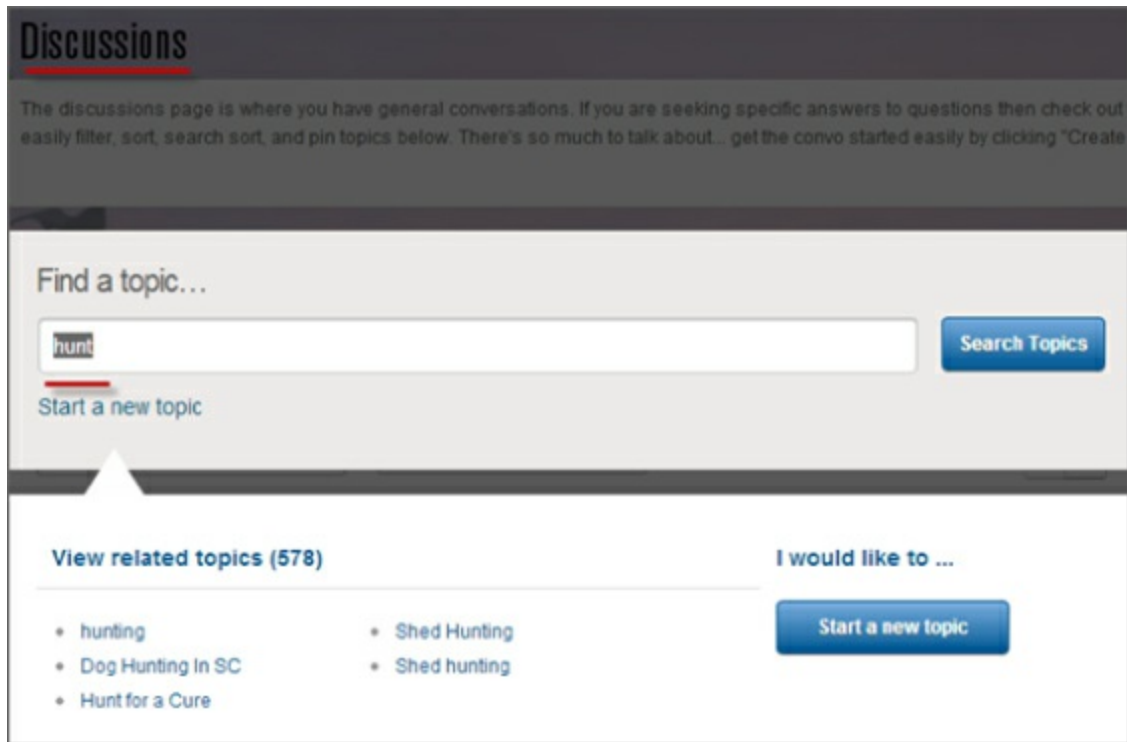
Although the DNN Platform supports the creation of more than one site in a given installation, search is scoped to one site at a time. The principles behind this feature are the same as those of the very sites themselves, where a given site is not supposed to know about the other sites due to security reasons. The Evoq products, however, have a feature called Site Groups that allows one site to be designated as a parent site and others to be child sites. In such a scenario, search can also be extended to cover the child sites. As shown in [Figure 11.8](#), administrators can select whether to scope the search to the current site or extend it to the child sites as well.



[Figure 11.8](#)

Module-Scoped Search

Results can also be scoped to specific modules. Module developers can easily integrate search into their modules; see the section “Module Integration” for more details. [Figure 11.9](#) shows an example of search integration in an Evoq product where the search is restricted to the Discussions module.



[Figure 11.9](#)

Advanced Search

Often the search skin object is sufficient for performing a basic search, given it has the auto-preview feature. However, advanced search is available to further narrow down the search results by different criteria such as how recent the results are, tags, and so on. The advanced search can be accessed by going into the Search Results module and clicking the Advanced button on the right. [Figure 11.10](#) shows the Advanced button. The various advanced options are additive, meaning the user can narrow down results by a specific tag and limit them to a certain number of days.

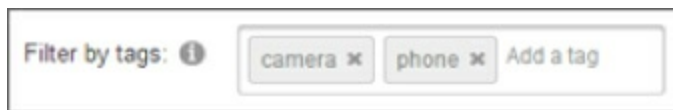


[Figure 11.10](#)

Filtering by Tags

One or more taxonomy tags (see [Figure 11.11](#)) can be specified to limit search results by tags. Many of the Evoq modules require users to supply a tag while

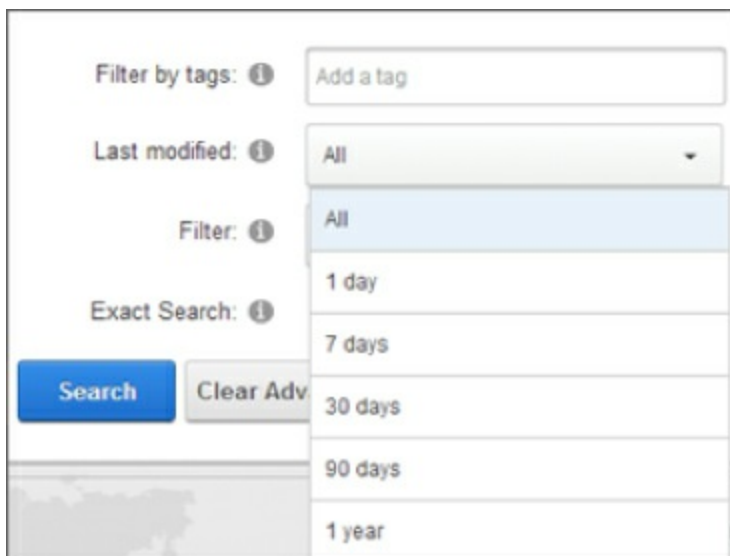
creating content (for example, a blog, a question, or an idea). This feature allows users to limit the search results to one or more tags across one or more modules.



[Figure 11.11](#)

Filtering by Content's Modification Time

Results can be restricted based on content's modification time. There are several options available out of the box (see [Figure 11.12](#)). This feature becomes handy when a user wants to search content created within a specified number of days. Users often have a vague idea when the content they are searching for was created; playing with this option helps them easily find their content.



[Figure 11.12](#)

Filtering by Search Types

Results can be restricted by search types. The Filter drop-down in [Figure 11.13](#) shows the list of such items. The content in the search index is organized by search type (see the section “Entities” for more details on `SearchType`). Within the Search Type module, the content is further organized by Module Definition ID.

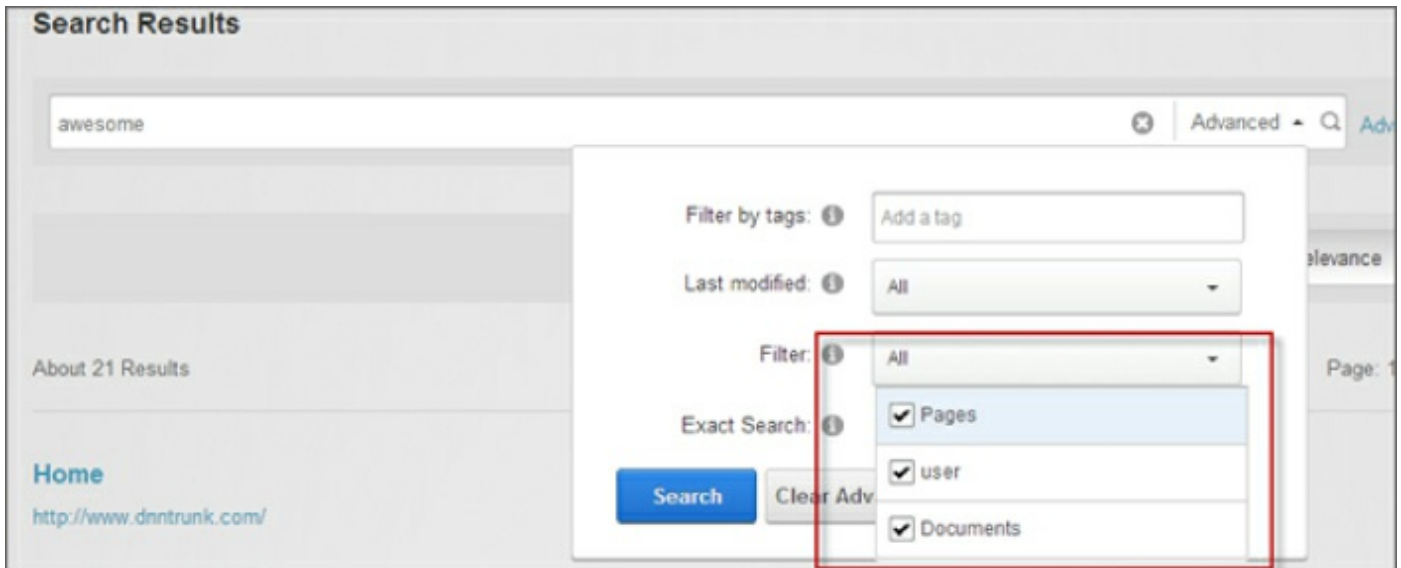


Figure 11.13

Each module that implements `ISearchable` or `ModuleSearchBase` is shown in this list for filtering. The HTML module's content is bundled in the Pages item, which also includes the page's name, its metadata, its module name, and its metadata.

The items under the Filter drop-down can be reduced through the module settings of the Search Results module. This provides control to administrators to hide the content of certain modules from searching. Users can further reduce the scope by unselecting a few more modules from the list.

Locale-Aware Search

Recent editions of the DNN Platform have made internalization and localization a high priority. There are about six language packs available out of the box starting with version 6.2. The current search functionality has been written with that in mind. Content stored in the search index has a culture code defined. This allows easy segregation of search results based on the current culture of a page.

As an example, if a site utilizes content localization and has made the site available in three languages—English, French, and Spanish—a French user will likely be viewing pages in the French language and will be interested in seeing search results from French pages only. The localization feature of Search is designed very well to handle this requirement. The upfront segregation of content in the index based on culture makes it very fast and easy to limit results to a specific language. One thing to note is that the

culture-neutral pages can be found for any language.

Delta Indexing

Search includes the concept of *deltas*, which allows modules to return only the changed content since the last time the platform probed it for content. This makes the system very efficient for indexing and reduces the burden on the server.

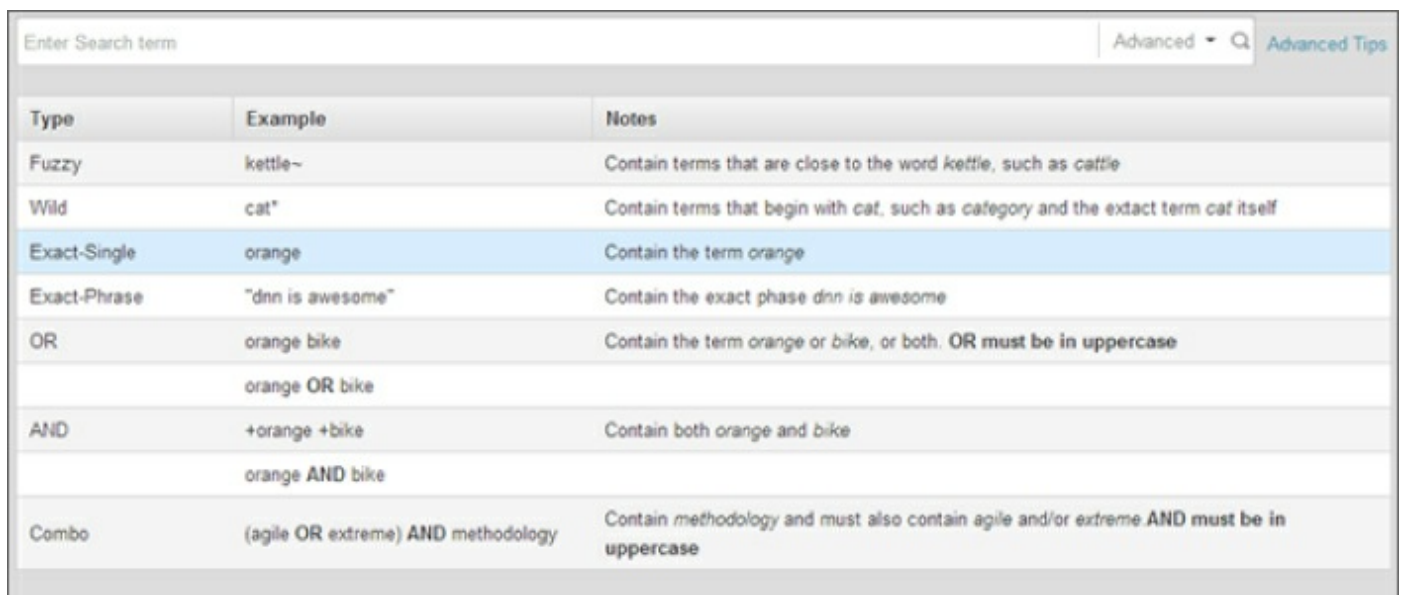
Ranking

Lucene is well-known for providing ranking of search results based on its own algorithm. Lucene gives higher ranks to the keywords that are used more frequently. It also considers the distance between different words to find the relevance of one keyword with respect to the other. These are just two examples of how Lucene does its ranking; there are other factors also. Module developers usually do not need to understand any of those.

While Lucene does its best to rank content, it also provides APIs that can increase ranking of specific content. See the section “SearchDocument” to learn different ranking options.

Lucene's Native Syntax

Lucene has its own native syntax to perform queries, which includes Boolean search, fuzzy search, wildcard search, and so on. A good portion of that syntax is available in DNN as well. [Figure 11.14](#) shows examples of how to create queries using Lucene's native syntax.



The image shows a search interface with a search bar at the top containing the text "Enter Search term". To the right of the search bar are two buttons: "Advanced" with a dropdown arrow and a magnifying glass icon, and "Advanced Tips". Below the search bar is a table with three columns: "Type", "Example", and "Notes".

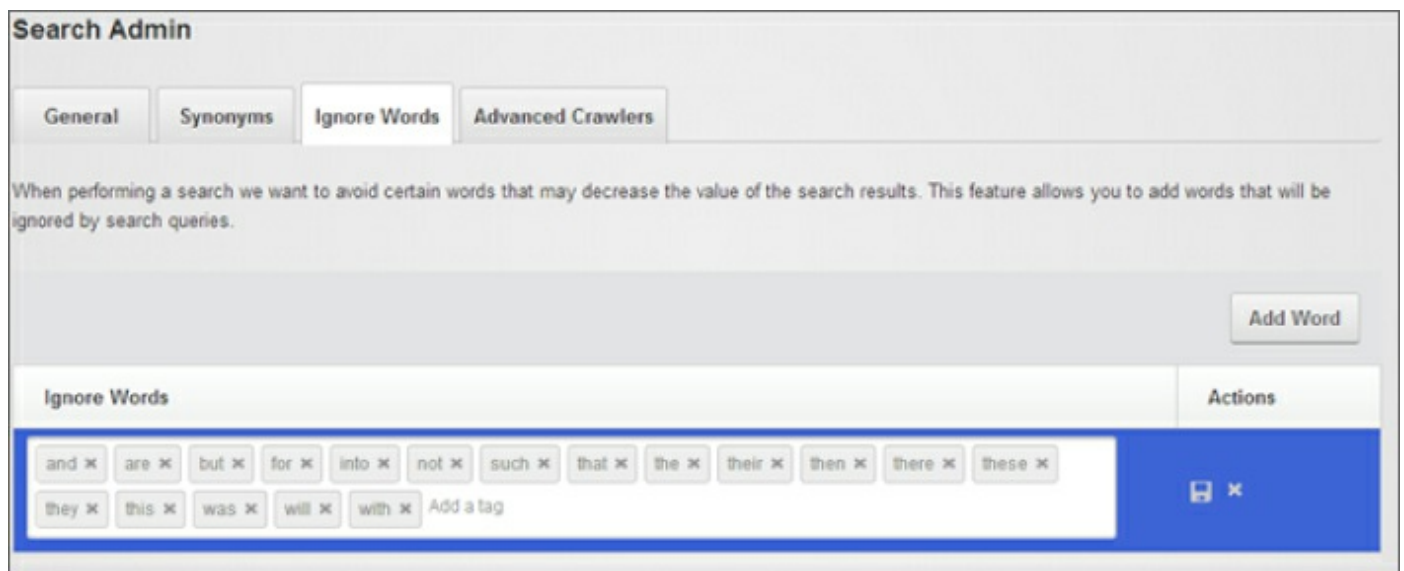
Type	Example	Notes
Fuzzy	kettle~	Contain terms that are close to the word kettle, such as cattle
Wild	cat*	Contain terms that begin with cat, such as category and the exact term cat itself
Exact-Single	orange	Contain the term orange
Exact-Phrase	"dnn is awesome"	Contain the exact phrase dnn is awesome
OR	orange bike orange OR bike	Contain the term orange or bike, or both. OR must be in uppercase
AND	+orange +bike orange AND bike	Contain both orange and bike
Combo	(agile OR extreme) AND methodology	Contain methodology and must also contain agile and/or extreme AND must be in uppercase

[Figure 11.14](#)

Fuzzy search is an interesting example. Fuzzy search can be executed by providing a tilde (~) after the keyword. For example, typing “kountry~” will also find “country”. Lucene applies a unique algorithm to treat words that sound similar as if they were the same word and uses that information to handle fuzzy searches.

Ignore Words

You can have Lucene ignore words that you don't want it to index even though modules or other sources send them for indexing. These words can be the typical stop words such as “a”, “the”, “not”, and so on, or they can be your competitor's name or just plain profane words that you don't want to index. DNN provides the standard English stop words in the list. The list can be easily modified by going into Admin \hookrightarrow Search Admin \hookrightarrow Ignore Words. These words can be configured per site and per language. See [Figure 11.15](#) for examples.



[Figure 11.15](#)

Synonyms

Synonyms refer to two or more words that have similar meanings, such as “understand” and “comprehend”. Lucene provides a way for administrators to introduce a group of words that mean the same thing. This can be done by going into Admin \hookrightarrow Search Admin \hookrightarrow Synonyms. See [Figure 11.16](#) for examples. In this case, searching for the word “DNN” will find not only “DNN” but also

“DotNetNuke” and vice versa because they are configured as synonyms.

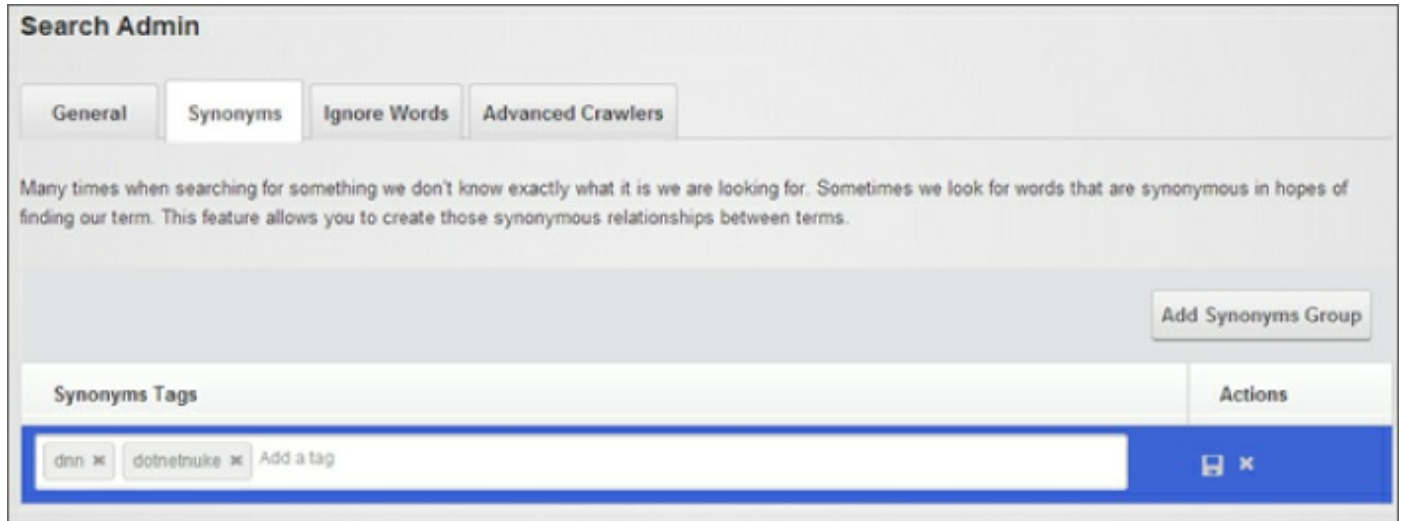
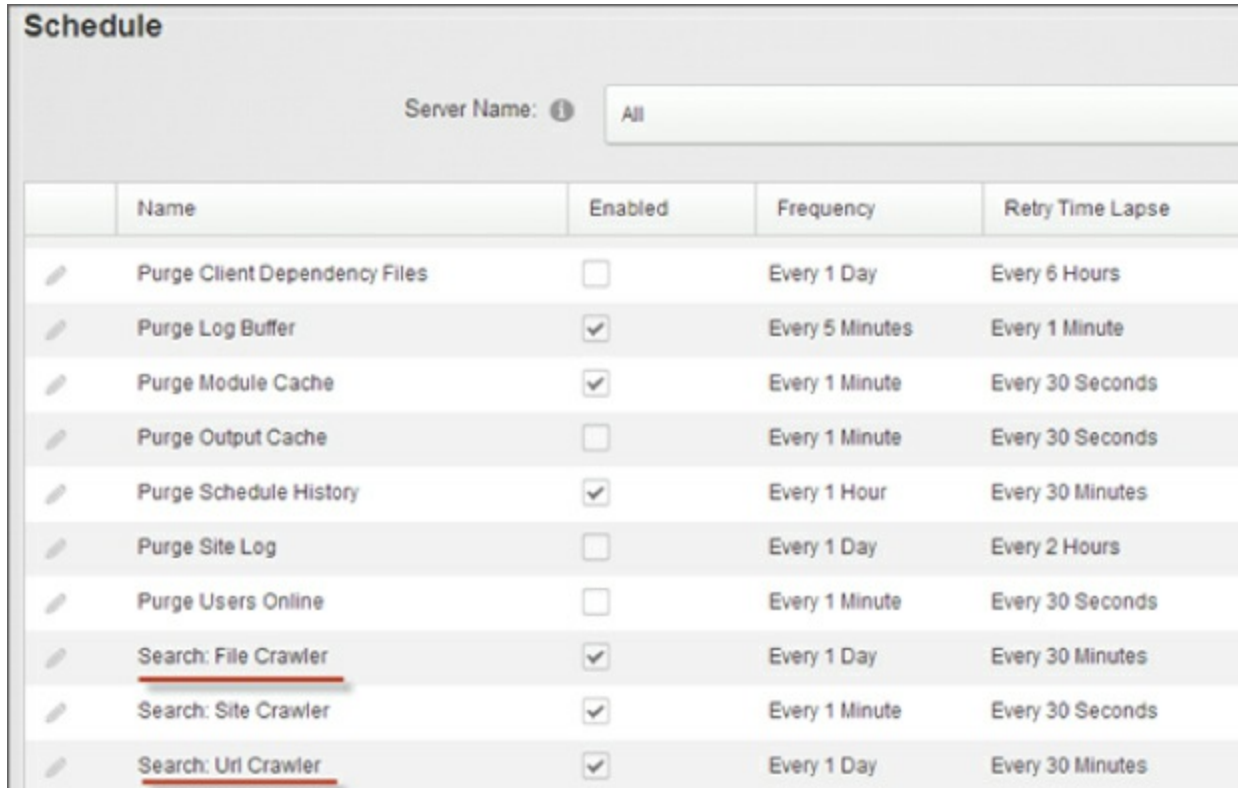


Figure 11.16

Evoq Features

Evoq products have two additional crawlers, which are URL and File Crawlers. Both of these crawlers have been part of the commercial offerings from the very beginning; however, they have been modified to take advantage of the new architecture. Similar to the platform's Site Crawler, both URL and File Crawlers are configured as scheduled tasks. [Figure 11.17](#) shows the URL and File Crawlers' scheduled frequencies.



	Name	Enabled	Frequency	Retry Time Lapse
	Purge Client Dependency Files	<input type="checkbox"/>	Every 1 Day	Every 6 Hours
	Purge Log Buffer	<input checked="" type="checkbox"/>	Every 5 Minutes	Every 1 Minute
	Purge Module Cache	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds
	Purge Output Cache	<input type="checkbox"/>	Every 1 Minute	Every 30 Seconds
	Purge Schedule History	<input checked="" type="checkbox"/>	Every 1 Hour	Every 30 Minutes
	Purge Site Log	<input type="checkbox"/>	Every 1 Day	Every 2 Hours
	Purge Users Online	<input type="checkbox"/>	Every 1 Minute	Every 30 Seconds
	<u>Search: File Crawler</u>	<input checked="" type="checkbox"/>	Every 1 Day	Every 30 Minutes
	Search: Site Crawler	<input checked="" type="checkbox"/>	Every 1 Minute	Every 30 Seconds
	<u>Search: Url Crawler</u>	<input checked="" type="checkbox"/>	Every 1 Day	Every 30 Minutes

[Figure 11.17](#)

URL Crawler

As the name suggests, the URL Crawler is driven by starting with one URL and traversing from one link to another. The site administrator can configure one or more URLs to be crawled. The crawler downloads the content of a URL first and then parses the HTML to obtain text to be sent to search index and obtain links for further crawling. Crawling continues this way until no new links are found. [Figure 11.18](#) shows the main setting for URL Crawler.

Search Admin

General Synonyms Ignore Words **Advanced Crawlers**

Expand All

URL:

Sitemap URL:

Enable Spidering:

DNN Role Impersonation: None
 Administrators
 Registered Users
 Subscribers
 Translator (en-US)
 Unverified Users

Windows Authentication:

Windows Domain (optional):

Windows User Account (optional):

Windows User Password (optional):

Figure 11.18

Scoping Below a URL

Crawling is restricted to the URLs below the starting URL. For example, if the starting URL is <http://www.mysite.com/site1>, then crawling will be done to the following example URLs:

- <http://www.mysite.com/site1>
- <http://www.mysite.com/site1/page1>
- <http://www.mysite.com/site1/page2>
- <http://www.mysite.com/site1/page1/sub/page>

It is important to note that the all of the URLs listed above have /site1 as the root-level folder. As can be seen, none of the URLs listed here contain the folder /site1 in its name. As a result, crawling will not be done to them:

- <http://www.mysite.com>

- <http://www.mysite.com/page1>
- <http://www.mysite.com/site2>
- <http://www.mysite.com/site3>
- <http://www.mysite.com/site2/page1>

DNN Role Impersonation

DNN's security model supports various roles, and in turn pages can have access defined per these roles. One set of roles may have access to certain sets of pages, whereas other roles may have access to other pages. It is very important to understand this concept and configure the URL Crawler to traverse the site as a specific role.

While configuring a URL for the current site, the administrator can specify different levels of roles, which include None, Administrators, Registered Users, Subscribers, and so on. The setting allows limiting of pages to crawl. For example, the setting None limits crawling to the anonymous pages. Unregistered users can search for content located in those pages only. Likewise, the setting of Registered Users limits crawling to pages that are accessible to both anonymous users and registered users.

Windows Authentication

This option allows the crawling of URLs protected by Windows authentication such as Active Directory. This option is used mostly for external sites in the intranet that are protected by user's role in the network. It is also used when the given DNN site uses Windows authentication. Domain information including username and password must be supplied while using this option.

No Duplication with Site Crawler

URL Crawler is smart enough to exclude content already indexed by the Site Crawler. When Site Crawler is enabled on a site, URL Crawler discards content generated by modules that participate in Site Crawler (that is, the ones that implement either `ISearchable` or `ModuleSearch`).

There are additional settings (see [Figure 11.19](#)) to handle duplicate URLs that point to the same page. These settings are complicated regular expressions that are used to determine whether two URLs point to the same content.

Ideally, the default settings are enough to handle most of the duplicates.

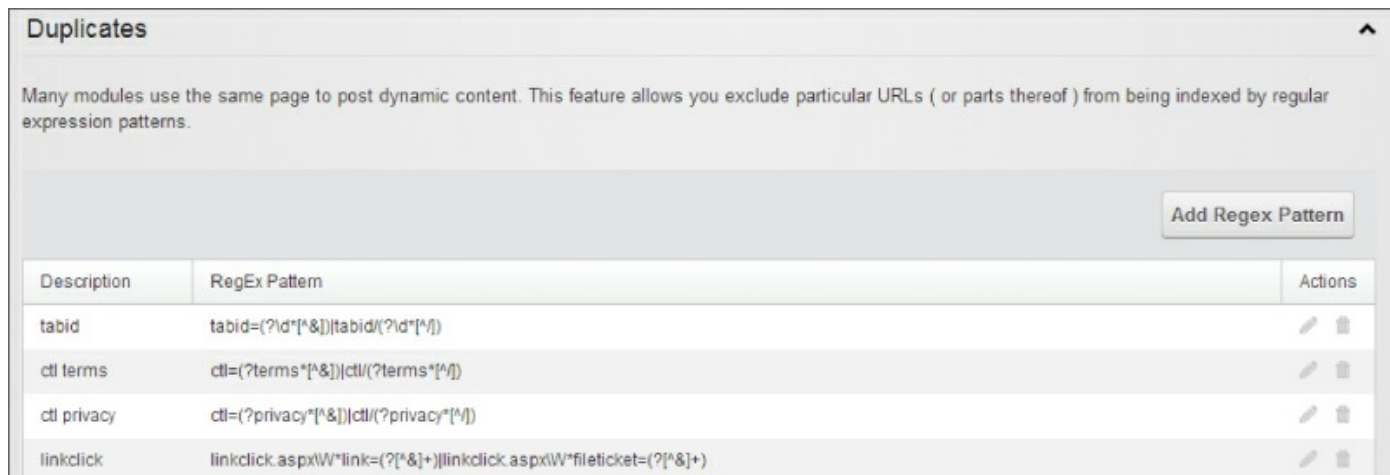


Figure 11.19

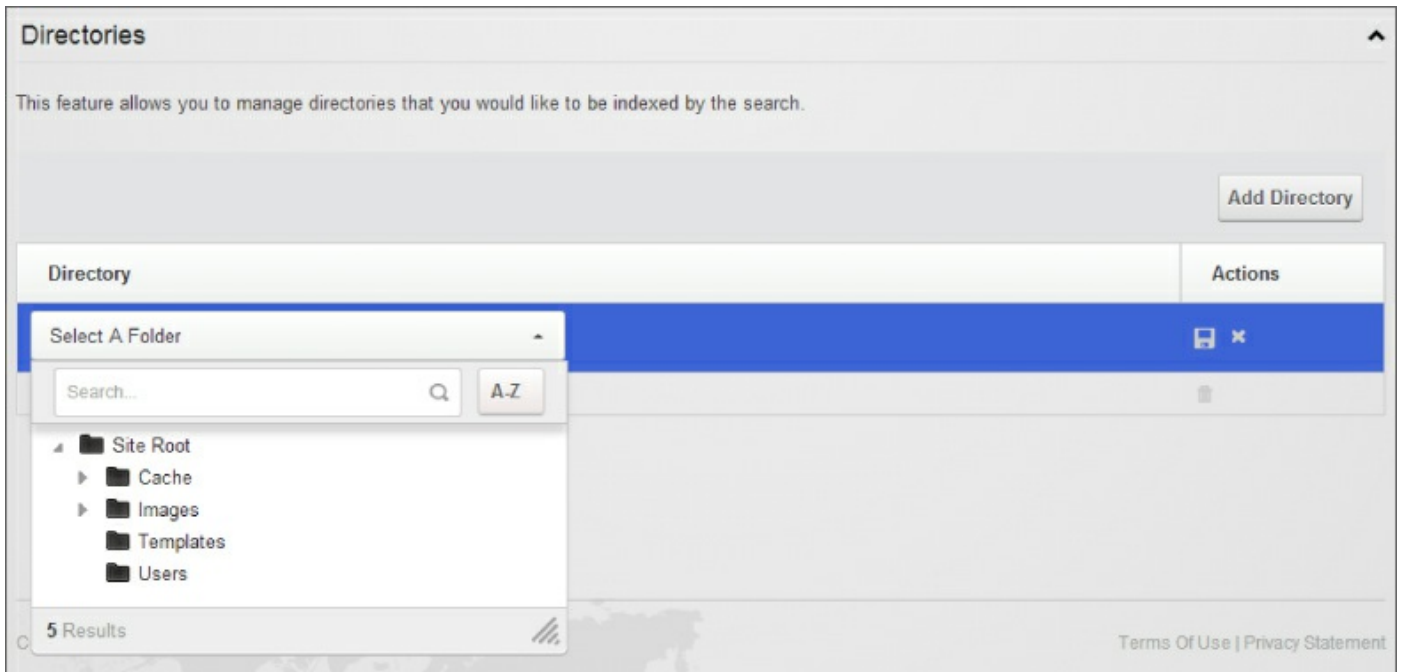
A good example is the tabid. A tabid in the query string such as tabid=99 or tabid/99 generally refers to the same URL and should thus be treated as a duplicate by URL Crawler.

File Crawler

File Crawler, as the name suggests, goes through files associated with Sites and Host and adds them to the index. In addition to indexing filenames and parent folders, a file's content is also indexed. Beyond that, you can also choose to exclude files.

Scoping by Folders

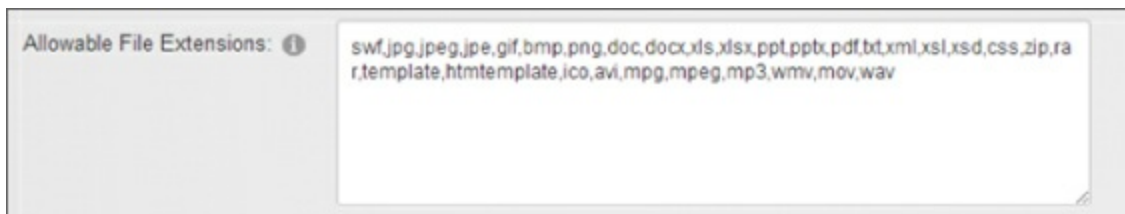
One or more folders can be specified (see [Figure 11.20](#)) for crawling. Subfolders are automatically included when a folder is selected. By default, the entire Site Root is selected.



[Figure 11.20](#)

Scoping by File Extension

File Crawler includes only those files that are defined under the Allowable File Extensions setting in Host \downarrow Host Settings \downarrow Other Settings (see [Figure 11.21](#)).

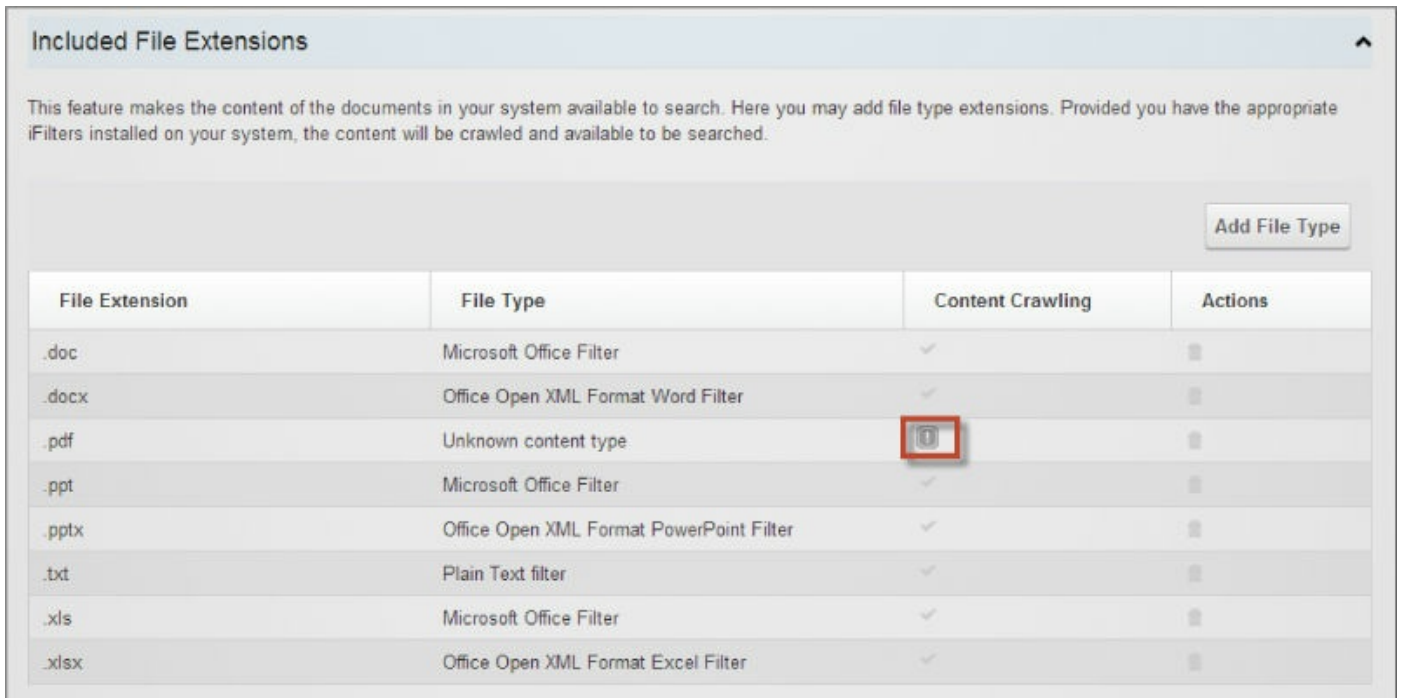


[Figure 11.21](#)

Crawling a File's Content

Along with indexing a file's metadata, File Crawler can also index the text portion of its content. Prior to 7.1, PDFBox and IKVM library for .Net were used for content checking. Starting in 7.1, this has been replaced by iFilters. An IFilter is a generic way to allow Windows's search engine to extract content from different file types. There are specific iFilters for 32-bit and 64-bit versions of Windows. The main purpose of iFilter is to extract textual data from files. For example, in order to index PDF files, an iFilter from Adobe must be installed.

The iFilter must be installed on the server where DNN is running. Microsoft's iFilter for Office documents is usually installed by default. The Advanced Crawlers section of the Search Admin UI can be easily used to find out whether an iFilter is installed on the server. The Content Crawling column in the UI has a tick mark for a given file extension to indicate that the iFilter is installed (see [Figure 11.22](#)).



Included File Extensions

This feature makes the content of the documents in your system available to search. Here you may add file type extensions. Provided you have the appropriate iFilters installed on your system, the content will be crawled and available to be searched.

Add File Type

File Extension	File Type	Content Crawling	Actions
.doc	Microsoft Office Filter	✓	⊞
.docx	Office Open XML Format Word Filter	✓	⊞
.pdf	Unknown content type	✓	⊞
.ppt	Microsoft Office Filter	✓	⊞
.pptx	Office Open XML Format PowerPoint Filter	✓	⊞
.txt	Plain Text filter	✓	⊞
.xls	Microsoft Office Filter	✓	⊞
.xlsx	Office Open XML Format Excel Filter	✓	⊞

[Figure 11.22](#)

Excluding by Extension

File extensions can also be added to be excluded from search functionality (see [Figure 11.23](#)). Although image files (.png, .jpg, and so on) are not part of the default settings, many customers exclude these extensions to avoid having a standard site's images and icons discovered by the search feature.

Excluded File Extensions ^

This feature allows you to specify file extensions for files that you do not want to be displayed in search results.

[Add File Type](#)













File Extension	Actions
.css	
.eot	
.htmtemplate	
.ico	
.rar	
.template	
.ttf	
.woff	
.xml	
.xsd	
.xsl	
.zip	

Figure 11.23

Administration

The most important aspect for search functionality that site administrators or super users need to know is the default location of the search files, the scheduled tasks, and the re-indexing option. When in doubt about something not getting indexed or found, it's often a good idea to delete the entire search folder, issue a re-index command, and start over. There are special considerations to be made when running a site in a web farm. Additional configuration may be needed to perform search in languages with complex alphabets such as Chinese.

Default Index File Location

By default, the Lucene index files are stored in the `App_Data\Search` folder (see [Figure 11.24](#)). This path can be modified by changing a host setting in the database using SQL. This folder can easily grow large over a period of time; compacting can be used to periodically shrink the files.

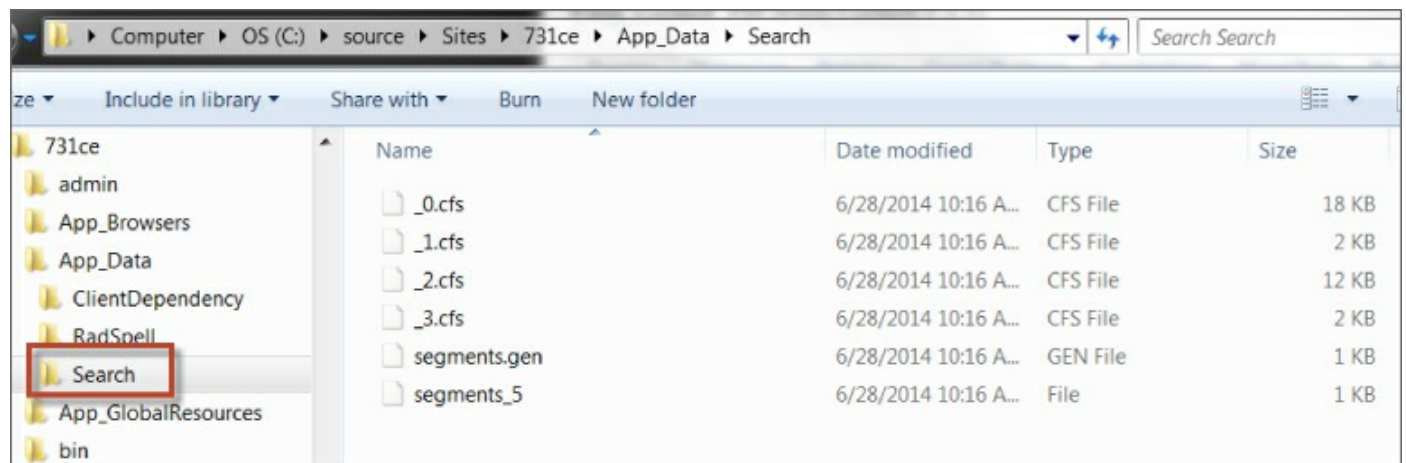


Figure 11.24

Web-Farm Configuration

While operating under a web farm, the search-related scheduled tasks (Site, URL, File, or any other search crawler) must run on just one web server. The setting can be set under Host ↕ Schedule ↕ Schedule Name ↕ Edit ↕ Run on Servers.

Lucene may run into locking issues when more than one process tries to write to the same index file. Although there is enough protection available in both the DNN and Lucene APIs to protect from locking within the same IIS

instance, there is no easy way to synchronize write operations across process or machine boundaries.

In any case, search-related scheduled tasks can run at the same time on the same web server.

Re-Indexing

There may be need to re-index the entire site. The platform provides re-indexing at site levels as well as at host level. While site-level re-indexing instructs crawlers to re-index just a site, host-level re-indexing re-indexes just the content specific to the host. It is important to note that the host content is searchable by super users only.

Re-indexing happens the next time crawlers are run. With regular indexing, the regular crawler gathers only changes in content since the last run. In contrast, the re-indexing start date for data gathering is set to the beginning of time.

If re-indexing is to be performed on all the sites and at the host level as well, it may be beneficial to delete the content of the App_Data\Search folder.

Depending on the size of the site, crawling after issuing re-indexing can cause CPU spikes, as it will be querying a potentially much larger amount of data.

Re-indexing at the site level is triggered by clicking the Re-Index Content button under Admin ⇨ Search Admin ⇨ General ⇨ Re-Index Content (see [Figure 11.25](#)).

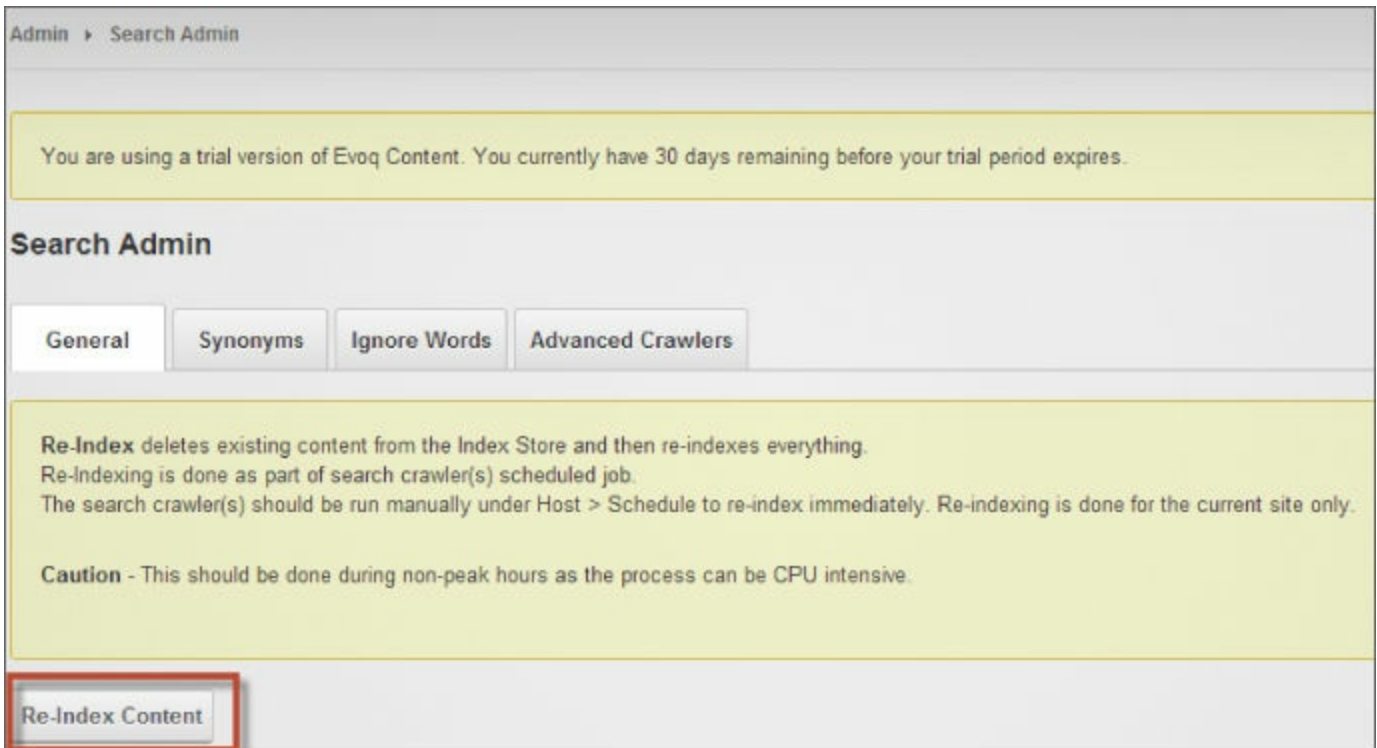


Figure 11.25

Reindexing at the host level is triggered by clicking the Re-Index Host Content button under Host \rhd Host Settings \rhd Advanced Settings \rhd Search Settings (see [Figure 11.26](#)).

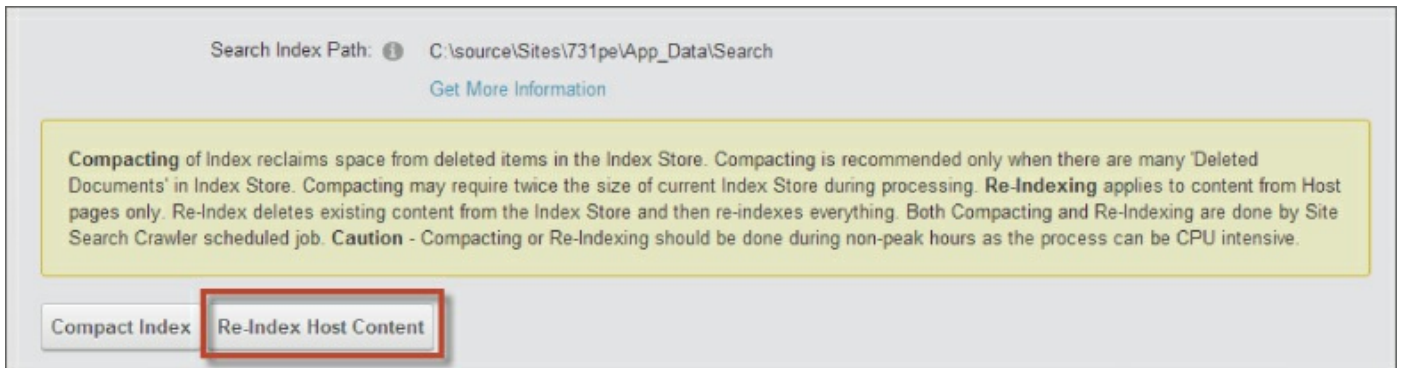
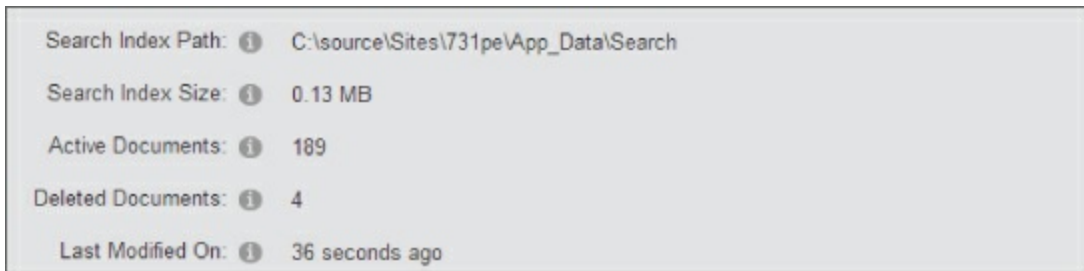


Figure 11.26

Getting Lucene Stats

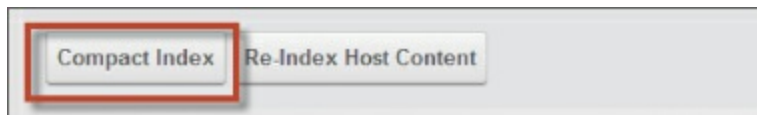
One way to gauge the size of Lucene's index is to take a look at the size of the App_Data\Search folder. The other way is to click the Get More Information link under Host \rhd Host Settings \rhd Advanced Settings \rhd Search Settings (see [Figure 11.27](#)). Information such as size, number of active or deleted documents, and last modified date is presented here.



[Figure 11.27](#)

Compacting

When a document is deleted from Lucene, it is not physically removed from the database. Instead, a flag is set that the document is deleted so it is not referenced in querying. To reclaim the space of deleted documents from Lucene, click the Compact Index button under Host \rhd Host Settings \rhd Advanced Settings \rhd Search Settings (see [Figure 11.28](#)). The compacting is done the next time the Site Crawler scheduled task runs.



[Figure 11.28](#)

Custom Analyzer

By default, Lucene uses the standard analyzer to perform text analysis on the content. The platform ships with more advanced analyzers, which are usually built on top of the standard analyzer. One common use-case is the Lucene.Net.Analysis.Cn.ChieneAnalyzer, which is used on sites in the Chinese language. A custom analyzer can be set by selecting Custom Analyzer Type under Host \rhd Host Settings \rhd Advanced Settings \rhd Search Settings (see [Figure 11.29](#)).

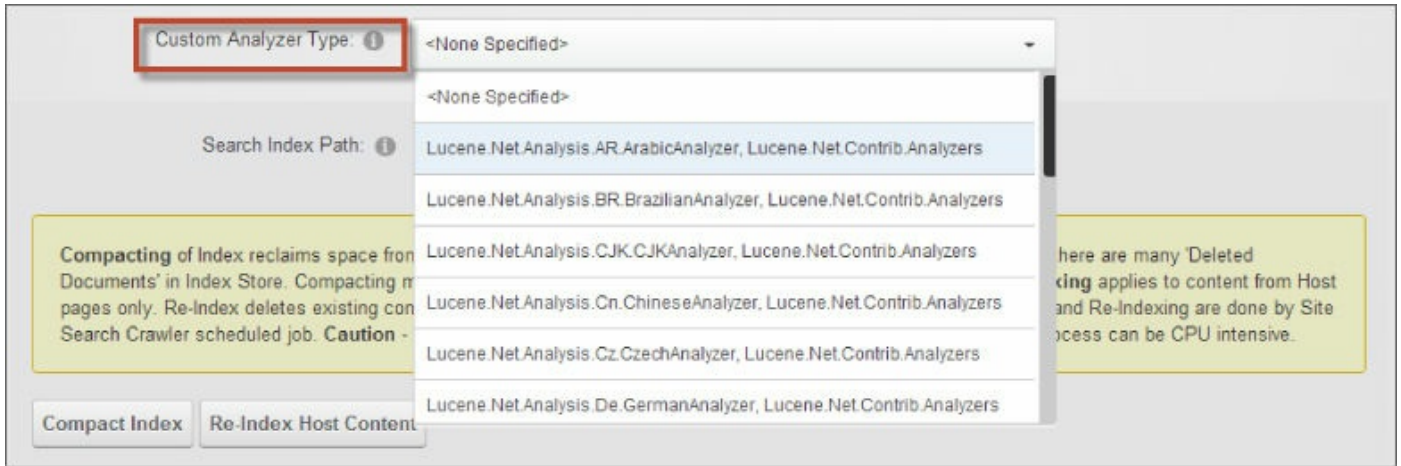
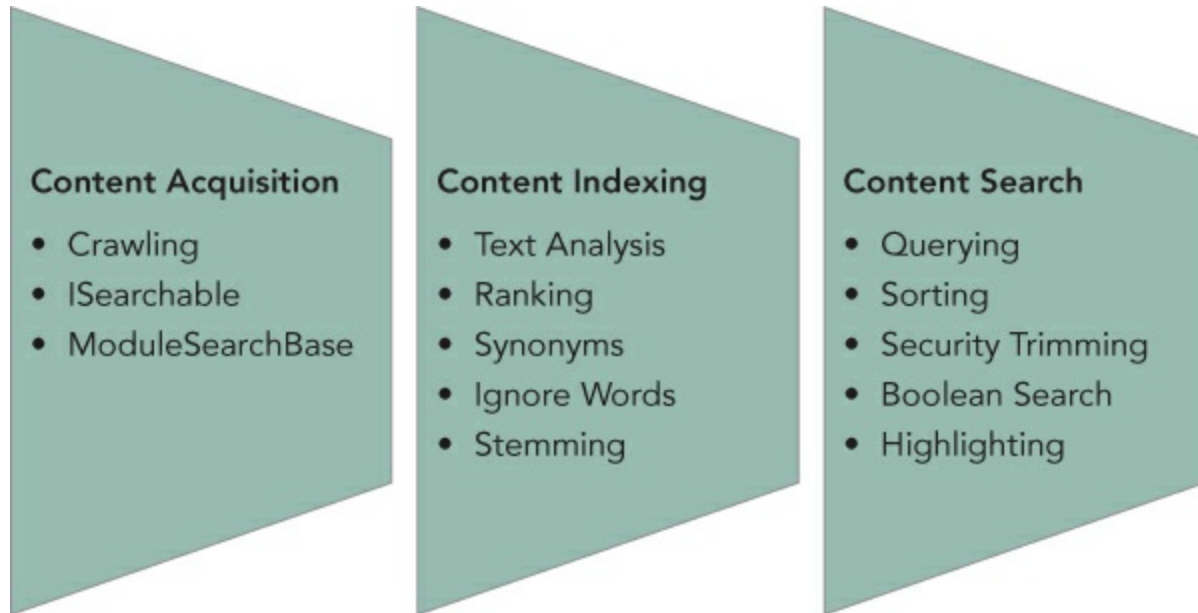


Figure 11.29

Search Phases

Search in DNN consists of three phases: content acquisition, content indexing, and content searching (see [Figure 11.30](#)). Content acquisition happens first and refers to the gathering of content from various sources. Content indexing is the second phase where acquired content is processed and stored in the Lucene store. The content search phase is the last step where content is searched by the end users.



[Figure 11.30](#)

Content Acquisition

The goal of content acquisition is to crawl various content sources and generate `SearchDocuments` to be stored in the index. The `SearchDocuments` are generated by the crawlers such as Site, File, and URL Crawlers. The Site Crawler in the platform probes modules implementing the `ISearchable` interface or `ModuleSearchBase` abstract class to acquire their content and store it in the Lucene store.

Content Indexing

The content indexing stage is responsible for performing text analysis, ranking, applying of synonyms, ignore words, stemming, and so on. All of these activities are performed using Lucene APIs. [Table 11.2](#) describes the steps involved in content indexing. All of these steps are performed one by one in the platform. Module developers do not have to specifically implement

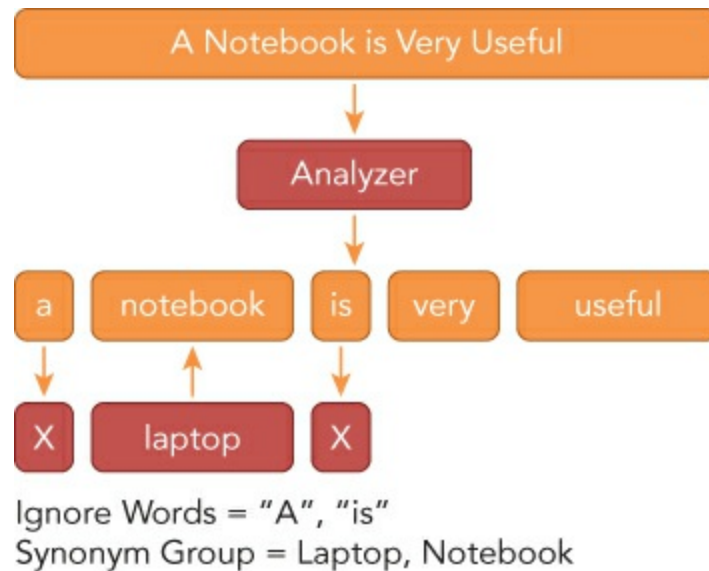
any of these steps. However, is helpful to understand how content gets transformed into Lucene objects. As described in the “Troubleshooting” section, Luke (a Java-based standalone tool) can be used to see more such details.

Table 11.2 Steps in Content Indexing

Step	Description
Text Analysis	<p>Text analysis breaks a sentence into individual words. There are various analyzers available in Lucene to perform this task. More than one analyzer can be used in a daisy-chain form. The following analyzers are used in the platform:</p> <ul style="list-style-type: none"> • Standard: Identifies email, acronyms, and so on. • Length: Skips words that are shorter than two characters. • Lowercase: Converts to lowercase. • StopFilter: Removes Ignore Words. • SynonymFilter: Applies synonyms. • ASCIIFoldingFilter: Removes accents from non-English words.
Ranking	Lucene uses Boost for ranking content. Documents as well as fields can be boosted higher.
Synonyms	Synonyms are words that mean same thing; for example, “notebook” and “laptop”. Ideally, a search for “notebook” should also find “laptop”. Another use is abbreviations; for example, a search for “DotNetNuke” should also find “DNN” and vice versa. Lucene places synonyms at the same position as the original word.
Ignore Words	Ignored Words are removed from the index. The platform ships with 20 standard English words to be ignored such as “a”, “an”, “and”, “the”, and so on. These words are removed while indexing.
Stemming	Stemming converts a word to its root and stores just the root in the index. For example “breathe”, “breathes”, “breathing”, and “breathed” are all stored as “breath”. Searching for one will find the others. DNN uses PorterStemFilter to perform stemming. There are also other stemming filters available in Lucene. Tests

indicated Porter to be better suited for non-English stemming.

[Figure 11.31](#) shows how a statement is first broken down into individual words and then the ignored words (for example, “a” and “is”) are removed. It also shows how synonyms (for example, “notebook” is the same as “laptop”) are applied.



[Figure 11.31](#)

Content Searching

Content searching is the last phase of searching, where users actually perform searches. All the standard Lucene boolean syntax is supported. Sorting can be performed based on relevance or date. Filtering can also be applied based on `SearchType` or module. Two important aspects of content searching are highlighting and security trimming:

- **Highlighting:** The output from querying consists of a snippet of the content where it's found. The hit is highlighted in the result, also.
- **Security trimming:** It is very important for the querying phase to return only those results to which a caller has access. Standard page and module permissions are applied on the results to trim out access-restricted items prior to returning back to the caller. Modules can have custom permissions applied on top of standard DNN permissions.

Module Integration

Modules can easily integrate with search functionality by implementing the `ModuleSearchBase` abstract class in their `BusinessController` class. The main purpose of this abstract class is to help modules easily send their new content for indexing into the Lucene store. Once content is added to the Lucene store, it can be searched by users through the search skin object or search results module. This is a simple use case where the module is interested in simply providing content to Lucene. In addition to this, module developers can also provide custom querying functionality in their modules by taking advantage of the `SiteSearch` and `ModuleSearch` APIs.

ModuleSearchBase

The Search in 7.1 introduced a new abstract class, `ModuleSearchBase`, to help modules easily integrate with search. The old `ISearchable` interface has been marked as obsolete, although it still works. `ModuleSearchBase` is more efficient as it uses the concept of deltas. `ISearchable` required modules to provide all their content all the time. `ModuleSearchBase` only asks for a difference in content since the last run.

`ModuleSearchBase` should be implemented in the `BusinessControllerClass` in the module's manifest. [Figure 11.32](#) shows a portion of a manifest from the HTML module. As always, you must provide `Searchable` as one of the `SupportedFeatures`.

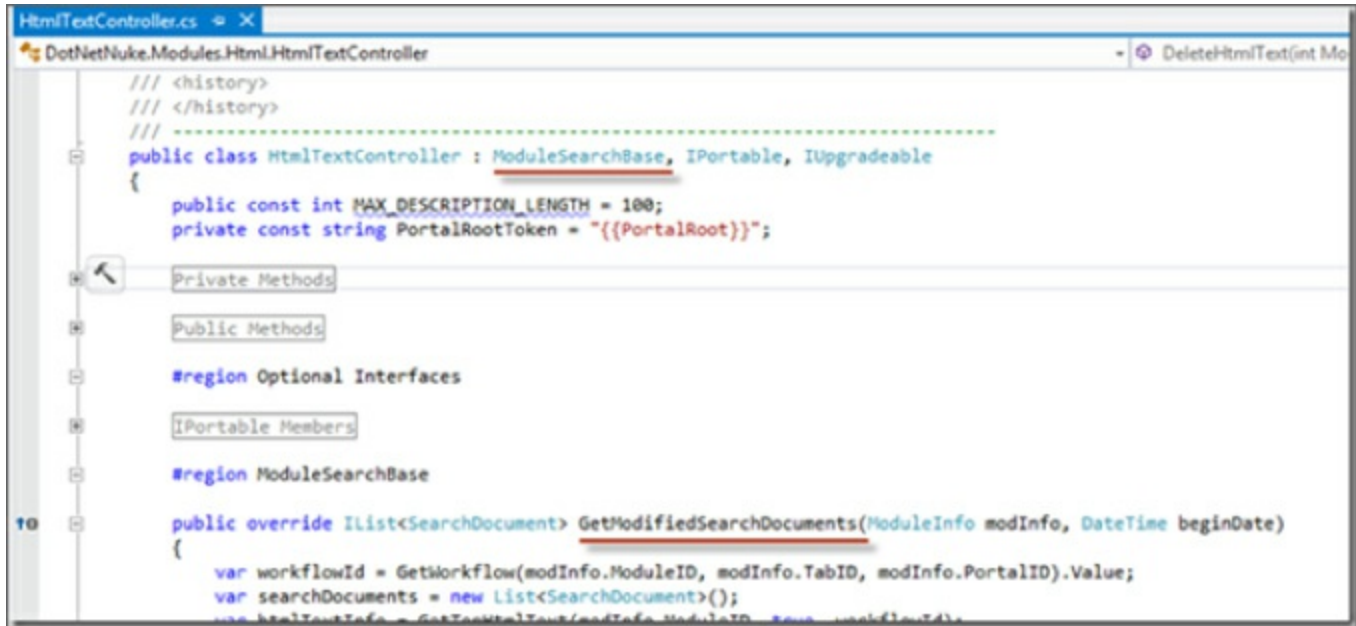


```
dnm_HTML.dnn  X
</component>
<component type="Module">
  <desktopModule>
    <moduleName>DNM_HTML</moduleName>
    <folderName>HTML</folderName>
    <shareable>Supported</shareable>
    <businessControllerClass>DotNetNuke.Modules.Html.HtmlTextController, DotNetNuke.Modules.Html</businessControllerClass>
    <supportedFeatures>
      <supportedFeature type="Portable" />
      <supportedFeature type="Searchable" />
      <supportedFeature type="Upgradeable" />
    </supportedFeatures>
  </component>
```

Figure 11.32

[Figure 11.33](#) shows the `ModuleSearchBase` implementation in the HTML module. As can be seen, the module implements a `GetModifiedSearchDocuments` method that returns a list of `SearchDocument` objects. This method in turn calls into its own controller methods to query content that has been changed since the `beginDate` was passed as a parameter. The parameter `modInfo` is used to pinpoint to the correct module

for its content.



```
HtmlTextController.cs
DotNetNuke.Modules.Html.HtmlTextController
DeleteHtmlText(int Mo

/// <history>
/// </history>
/// -----
public class HtmlTextController : ModuleSearchBase, IPortable, IUpgradeable
{
    public const int MAX_DESCRIPTION_LENGTH = 100;
    private const string PortalRootToken = "{{PortalRoot}}";

    Private Methods
    Public Methods

    #region Optional Interfaces
    IPortable Members

    #region ModuleSearchBase

    10 public override IList<SearchDocument> GetModifiedSearchDocuments(ModuleInfo modInfo, DateTime beginDate)
    {
        var workflowId = GetWorkflow(modInfo.ModuleID, modInfo.TabID, modInfo.PortalID).Value;
        var searchDocuments = new List<SearchDocument>();
        var htmlTextInfo = GetHtmlText(modInfo.ModuleID, modInfo.TabID, workflowId);
```

Figure 11.33

GetModifiedSearchDocuments

`GetModifiedSearchDocuments` is the only method in `ModuleSearch`. Modules participating in a search should implement this method and return `SearchDocuments` for new, changed, and deleted content in the module. This method accepts `ModuleInfo` and `BeginDate` as parameters. The `BeginDate` is in the UTC format.

The `GetModifiedSearchDocuments` method is called periodically by Site Crawler, and it is called for each and every module instance that implements either `ISearchable` or `ModuleSearchBase`. There is some intelligence built into the Site Crawler so that it calls only those modules whose `LastContentModifiedOnDate` property from the `ModuleInfo` object is later than the last crawler run date time. Not all modules implement this behavior. As of this writing, only the HTML module updates its `LastContentModifiedOnDate`. This helps a lot from a performance point of view as the HTML module is the most commonly used module on sites.

You need to be careful with packages that have more than one module definition in the manifest having `SupportedFeature` as `Searchable`. In such situations, this will be called for all module definitions defined in that manifest. In most of these modules (for example, the Blogs module), there is usually one main module with other helper modules.

The module should inspect the supplied `ModuleInfo` object to ensure that it points to the main module definition and then only return content; otherwise it should return an empty collection. Not doing this may result in duplicate content being indexed. The old `ISearchable` worked this way as well.

Entities

The new search functionality primarily defines four entities—`SearchType`, `SearchDocument`, `SearchResult`, and `SearchQuery`. [Figure 11.34](#) shows the places where these entities are used. `SearchType` is used to distinguish the different types of content stored in Lucene, such as content from modules, metadata about pages, information about users, and so on. `SearchDocument` is the general-purpose object to be used to send a piece of content for indexing into Lucene.

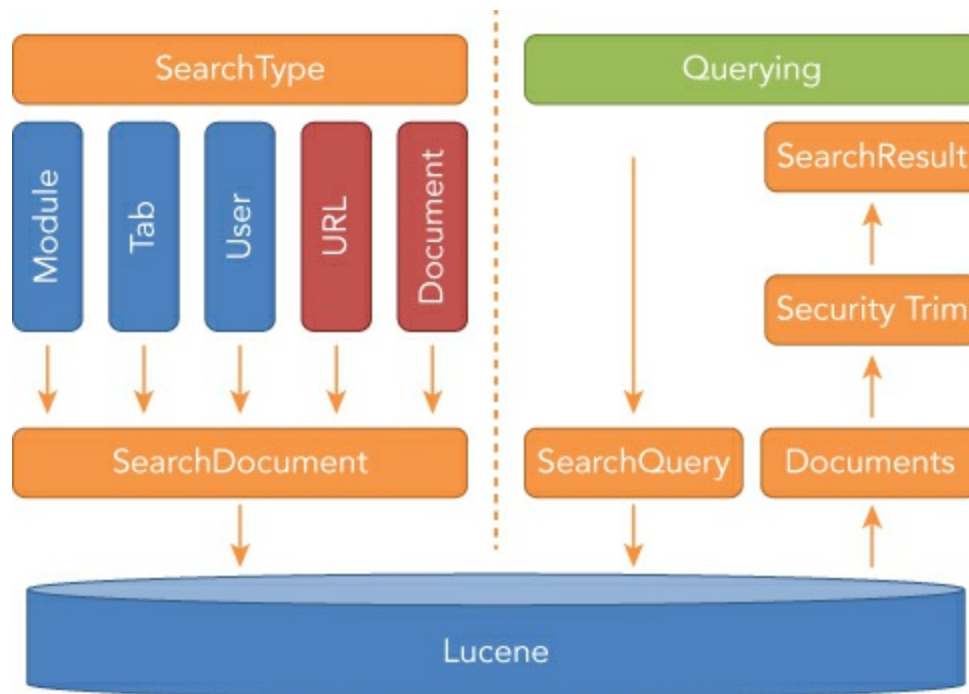


Figure 11.34

`SearchQuery` and `SearchResult` are used in the querying part of Search. `SearchQuery` is the object to build a simple or complex query to run against Lucene, and `SearchResult` is the object that is returned from Lucene once the query is executed. The three DNN entities `SearchDocument`, `SearchQuery`, and `SearchResult` correspond to Lucene's native `Document`, `Query`, and `ScoreDoc` entities, respectively. There is no equivalent of `SearchType` in Lucene; it's a DNN platform-specific entity.

SearchType

Search in DNN is written with extensibility in mind. `SearchType` provides developers the ability to control certain aspects of the search results—primarily custom permissions and custom URLs. Most module developers do

not need to work with `SearchType`. However, `SearchType` is handy when new crawlers are to be written or default behavior around search permission or URL needs to be changed in the platform.

The DNN platform ships with Site Crawler, whereas Evoq products ship with URL and File Crawlers. All three crawlers are implemented as scheduled tasks, and each has one or more search types. For example, the Site Crawler in the DNN platform has three: module, tab, and user. The URL and File Crawlers define one `SearchType` each. [Table 11.3](#) explains the `SearchType` properties.

Table 11.3 `SearchType` Properties

Property	Description
<code>SearchTypeId</code>	Search Type ID.
<code>SearchTypeName</code>	Search type name.
<code>SearchResultClass</code>	A class implementing <code>BaseResultController</code> . This class will be invoked by reflection.
<code>IsPrivate</code>	Content from this <code>SearchType</code> will normally not be searched while performing site or module search.

SearchDocument

`SearchDocument` is the object that is transformed into a native Lucene document and is stored in its NoSQL database. The properties from `SearchDocument` are transformed into native Lucene fields. [Table 11.4](#) explains the `SearchDocument` properties.

Table 11.4 `SearchDocument` Properties

Property	Description
<code>UniqueKey</code>	A key to uniquely identify a document in the index.
<code>Title</code>	The content's title. HTML tags are stripped from this property, but certain HTML attribute values will be retained, such as <code>alt</code> and <code>title</code> attribute values.
<code>Description</code>	The content's description. The description should generally be no more than two sentences. This property is used by RSS syndication. It is also used in search results when highlighted text is not found during searching. HTML tags

	are stripped from this property, but certain HTML attribute values are retained, such as <code>alt</code> and <code>title</code> attribute values.
<code>RoleId</code>	<code>RoleId</code> (<code>GroupId</code>) is for additional filtering. This field is optional. This property can be used while under social groups.
<code>Body</code>	The content's body. HTML tags are stripped from this property, but certain HTML attribute values will be retained, such as <code>alt</code> and <code>title</code> attribute values.
<code>Url</code>	The URL for the indexed item. Usually <code>TabId</code> or <code>ModuleId</code> is enough to generate a document URL in the search result. However, the <code>Url</code> field is used if present in <code>SearchResult</code> .
<code>PortalId</code>	The ID of the Portal.
<code>TabId</code>	The Tab ID of the content. This field is optional.
<code>ModuleDefId</code>	The Module Definition ID of the content. This is needed when <code>SearchTypeId</code> is for a module.
<code>ModuleId</code>	The Module ID of the content. This is needed when <code>SearchTypeId</code> is for a module.
<code>AuthorUserId</code>	The User ID of the author. The author's display name is automatically found and stored. <code>AuthorName</code> can be found in <code>SearchResult</code> . However, this may be out of date if <code>DisplayName</code> is changed after indexing.
<code>SearchTypeId</code>	The Search Type ID; for example, module, file, or URL.
<code>ModifiedTimeUtc</code>	The time when content was last modified (in UTC).
<code>IsActive</code>	The flag to indicate whether content is active. Content will be deleted from the index when <code>IsActive</code> is <code>false</code> . The default is <code>True</code> .
<code>QueryString</code>	The query string that may be associated with a search document. This information will be used to create a URL for the document.
<code>Permissions</code>	A string representation of roles and users who have view (or are denied view) permissions.
<code>Keywords</code>	Additional keywords can be specified for indexing. This is key-value pair, for example, "AliasName," "something."
<code>NumericKeys</code>	Additional numeric fields can be specified for indexing.

	This is key-value pair; for example, “ItemId,” “888.”
Tags	Tags can be specified as additional information.
CultureCode	The culture code associated with the content.

The DNN Search APIs bump up the boost level of a few of the `SearchDocument` properties based on [Table 11.5](#). The remainder of the properties are assigned a boost of 1, meaning there is no bump in boost or ranking. The default boost levels can be changed by manually tweaking the host settings noted in the [Table 11.5](#). The value of the host setting should be multiplied by 10 compared to the boost level you want to change. For example, if you want to set the boost value of `Title` to be 7, then you must set the host setting `Search_Title_Boost` to be 70.

Table 11.5 Property Boost Levels

Property	Boost	Host Setting Name
Title	5	Search_Title_Boost
Tag	4	Search_Tag_Boost
Keyword	3.5	Search_Content_Boost
Description	2	Search_Description_Boost
Author	1.5	Search_Author_Boost
Default	1	n/a

SearchQuery

As the name suggests, the `SearchQuery` entity is used to construct a search query to retrieve content from Lucene. The properties of the object can be filled to narrow down results. [Table 11.6](#) explains the `SearchQuery` properties.

Table 11.6 SearchQuery Properties

Property	Description
UniqueKey	A key to uniquely identify a document in the index. This should be passed only when a very specific document is to be retrieved.
KeyWords	Keywords to search for. Boolean options such as AND or OR can also be specified.
PortalIds	A collection of Portal IDs of the site upon which to

	perform the search. This field must be specified or portal 0 will be searched by default.
SearchTypeIds	A collection of Search Type IDs that should be searched upon. This is optional.
ModuleDefIds	A collection of Module Definition IDs that should be searched upon. This is optional.
ModuleId	The Module ID to restrict the search to. Only a value greater than 0 is used.
RoleId	The Role ID to restrict the search to. Only a value greater than 0 is used. This property does not have any relationship with security.
TabId	The Tab ID to restrict the search to. Only a value greater than 0 is used.
Locale	The locale to which to restrict the search. This field can be left empty for a single language site. For example, a value of en-US or nl-NL can be specified to restrict search to a single locale.
BeginModifiedTimeUtc	The begin date of the time when content was last modified (in UTC). This field is optional.
EndModifiedTimeUtc	The end date of the time when content was last modified (in UTC). This field is optional.
Tags	This restricts search to specific tags. This field is optional.
PageIndex	The page index for the result: for example, pageIndex=1 and pageSize=10 indicates first 10 hits. The default value is 1.
PageSize	The page size of the search result. The default value is 10.
TitleSnippetLength	The maximum length of highlighted title field in the results.
BodySnippetLength	The maximum length of highlighted body snippet field in the results.
CultureCode	The culture code associated with the content. Culture-neutral content is always returned even though this value is specified.

SortField	The sort option of the search result (descending or ascending). This field is optional.
CustomSortField	The name of the custom sort field. This works with the <code>SortFields.CustomNumericField</code> or <code>SortFields.CustomStringField</code> option. Enum <code>SortFields</code> can be used to sort on <code>Relevance</code> , <code>LastModified</code> , and <code>Title</code> . Additional fields such as the ones provided under <code>SearchDocument.Keywords</code> , <code>SearchDocument.NumericKeys</code> , or <code>Tags</code> can be specified. It is important that the field name is a valid one.
WildcardSearch	Set this to <code>true</code> to perform a wildcard search. This property is not respected when keywords contain the special boolean phrases “~”, “*”, “\”, “\”, “and”, “or”, “+”, or “-”. When <code>WildcardSearch</code> is enabled, an additional OR is performed; for example <code>(keyword OR keyword*)</code> . It adds an asterisk at the end to find any words starting with the keyword. There can be performance implications with this setting turned on.

SearchResult

As the name suggests, the `SearchResult` entity is the object that is returned after performing a query. `SearchResult` inherits from `SearchDocument` and therefore contains all the properties defined under `SearchDocument`. `SearchResult` has the additional properties described in [Table 11.7](#).

Table 11.7 `SearchResult` Properties

Property	Description
<code>DisplayModifiedTime</code>	The time when content was last modified (in friendly format).
<code>Snippet</code>	The highlighted snippet from the document.
<code>AuthorName</code>	The display name of the author. This may be different from the current <code>DisplayName</code> when the index was run prior to the change in the <code>DisplayName</code> . This field is optional.

Score	Lucene's original score. This is the score of this document for the query. This field may not be reliable as most of the time it contains <code>Nan</code> . Use <code>DisplayScore</code> instead.
DisplayScore	Lucene's original score in string format; for example, <code>1.45678</code> or <code>0.87642</code> . This is the score of this document for the query. This field is more reliable than the float version of <code>Score</code> .

APIs

The majority of the module developers can simply implement `ModuleSearchBase` and integrate with the search functionality. However, there are additional APIs available for performing more advanced operations. Search has two public APIs. There are also a few internal APIs, but they can be called from modules.

The Evoq products make use of the two public APIs to provide module-specific querying capabilities. The APIs that work with `SearchDocument` such as `AddSearchDocument`, `AddSearchDocuments`, and `DeleteSearchDocument` are mostly used by the crawlers. The administrative APIs are used by the host settings section of the platform.

Public APIs

Due to limitations in the web farm support with regard to writing to the index, only the querying APIs have been made truly public. Others are technically public; however, they are placed under the `Internals` namespace for now. The public APIs include the following:

- **SiteSearch:** Executes a `SearchQuery` within the scope of a site
- **ModuleSearch:** Executes a `SearchQuery` within the scope of a module

Unsupported (yet Useful) APIs

Any write operation performed on the Lucene index is executed by the search crawlers, which run as scheduled tasks. The Site Crawler calls into APIs located under the `DotNetNuke.Services.Search.Internals` namespace to perform a write or delete operation on the index. [Table 11.8](#) describes the indexing APIs, [Table 11.9](#) describes the administration APIs, and [Table 11.10](#) describes the internal APIs.

[Table 11.8](#) Indexing APIs

API	Description
<code>AddSearchDocument</code>	Adds one <code>SearchDocument</code> into the index. The caller must call <code>Commit</code> to save the changes. <code>DeleteSearchDocument</code> is automatically called as well.

AddSearchDocuments	Adds a collection of SearchDocuments into the index. Commit is automatically called at the end. Lucene may occasionally commit internally as well if there are too many uncommitted documents in its internal buffer.
DeleteSearchDocument	Deletes SearchDocument from the index. Deletion is based on UniqueKey. ModuleDefintionId and ModuleId are also used when the SearchTypeId supplied is that for Module. Lucene does not remove the documents immediately; instead, it flags them for removal. The OptimizeSearchIndex API eventually clears them from the physical file.
DeleteSearchDocumentsByModule	Deletes all SearchDocuments related to a module.
DeleteAllDocuments	Deletes all SearchDocuments belonging to a specific SearchTypeId.
Commit	Commits changes to physical files in the index.

Table 11.9 Administration APIs

API	Description
GetSearchStatistics	Obtains index statistics such as index size, number of documents, and so on
OptimizeSearchIndex	Compacts the index by removing deleted documents

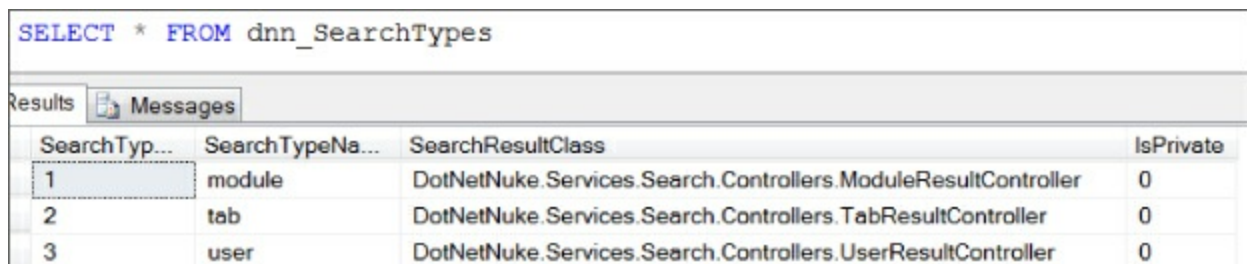
Table 11.10 Lucene Internal APIs

API	Description
Add	Adds a Lucene document into the index
Search	Performs search queries in Lucene

Writing a New Crawler

Any `SearchDocument` stored in Lucene must have `SearchType` associated with it. When a user performs a search query and a hit is subsequently found in the Lucene store, Lucene returns the entire `SearchDocument` back to the platform. The platform first looks at the `SearchType` present in the `SearchDocument` to find the corresponding `BaseResultController` from `SearchTypes` table. It then calls the `BaseResultController` to ensure that the user running the query has view permissions on the `SearchDocument` and that a custom URL is defined for the result.

Search can easily be extended to support different `SearchTypes`. As previously noted, the platform defines `module`, `tab`, and `user` `SearchTypes`. The Evoq products have `URL` and `Document`. [Figure 11.35](#) shows the standard `SearchTypes` defined in the `SearchTypes` table.



SearchTyp...	SearchTypeNa...	SearchResultClass	IsPrivate
1	module	DotNetNuke.Services.Search.Controllers.ModuleResultController	0
2	tab	DotNetNuke.Services.Search.Controllers.TabResultController	0
3	user	DotNetNuke.Services.Search.Controllers.UserResultController	0

Figure 11.35

The following are required to create a new crawler:

- Create a new `SearchType` and add it to `SearchTypes` table.
- Be sure to provide a meaningful `SearchTypeName`.
- You can provide a friendly name and further localize it by adding an entry in the `Website\DesktopModules\Admin\SearchResults\App_LocalResources\Se` resource file.
- The class name defined in the `SearchResultClass` column must inherit from the abstract class `BaseResultController`.
- Create a new scheduled task to call your crawler.
- From within the crawler, store data in the Lucene store by calling the `InternalSearchController.Instance.AddSearchDocuments` API.
- Once a search hit is found in the content associated with this new

SearchType, the platform will automatically call the `SearchResultClass` defined above for the purposes of custom permission or URL processing.

The `SearchResultClass` column in the `SearchTypes` table should point to the class inheriting from `BaseResultController`.

The `BaseResultController` has two abstract methods—`HasViewPermission` and `GetDocUrl`—that must be implemented. The `HasViewPermission` method helps in security trimming and `GetDocUrl` helps in generating custom (if needed) URLs for the result. [Figure 11.36](#) shows the outline of `BaseResultController`.

```
namespace DotNetNuke.Services.Search.Controllers
{
    /// <summary>
    /// BaseResult to be implemented by the different Crawlers to provide Permission and Url Services
    /// </summary>
    /// <remarks>The abstract methods in this Class will be called by Search Result engine for every Hit found in Search Index.</remarks>
    [Serializable]
    public abstract class BaseResultController
    {
        #region Abstract

        /// <summary>
        /// Does the user in the Context have View Permission on the Document
        /// </summary>
        /// <param name="searchResult">Search Result</param>
        /// <returns>True or False</returns>
        public abstract bool HasViewPermission(SearchResult searchResult);

        /// <summary>
        /// Return a Url that can be shown in search results.
        /// </summary>
        /// <param name="searchResult">Search Result</param>
        /// <returns>Url</returns>
        /// <remarks>The Query Strings in the Document (if present) should be appended while returning the Url</remarks>
        public abstract string GetDocUrl(SearchResult searchResult);

        #endregion
    }
}
```

Figure 11.36

The behavior of any of the existing built-in `BaseResultControllers` can also be changed by following these steps:

1. Create a new class that inherits from `BaseResultController`.
2. This class can be defined in any of your DLLs.
3. Update the `SearchResultClass` column in the `SearchTypes` table for the specific `SearchType` that you want to change behavior for.
4. Restart the site.

The next time a search hit is found for that `SearchType`, the platform will use your custom class as opposed to the built-in one for the purposes of custom

permission or URL processing.

Troubleshooting

Search is comprised of many areas such as indexing, storing in Lucene, querying, and so on. Problems can arise in any of these areas. To start with, modules must be found by the platform so that it can probe them for content. Once content is found, it must also be stored in the correct format in the Lucene store. Finally, the content should be searchable. In order to troubleshoot and see some diagnostic data, tracing can be enabled using the standard Log4Net config file.

Luke is a great tool for analyzing the content stored in the Lucene store. Developers should investigate Luke regardless of the need to troubleshoot search. Spending time with Luke will help understand how Lucene works and how data is stored in its database. It is interesting to know how your SQL-based data gets transformed into the platform's `SearchDocument` objects and is finally stored as NoSQL objects in Lucene.

Indexing

A common problem is when you have implemented `ModuleSearchBase` but your content is not getting indexed. There is a way to troubleshoot this situation. Run the following SQL to check whether your module is listed:

```
exec dnn_GetSearchModules 0
```

The number 0 is the Portal ID. [Figure 11.37](#) shows the standard output produced by running the previous command.

Owner	PortalID	TabID	TabModuleID	ModuleID	ModuleDefID	ModuleOrder	PaneName	ModuleTitle	CacheTime	CacheMethod	Alignment	Color	
1	0	0	55	51	362	116	1	ContentPane	Welcome to Your Installation	1200	FileModuleCachingProvider	NULL	NULL
2	0	0	56	52	363	116	1	sidebarPane	Sidebar	1200	FileModuleCachingProvider	NULL	NULL
3	0	0	56	53	364	116	1	footerRightPane	Contact Us	1200	FileModuleCachingProvider	NULL	NULL
4	0	0	56	54	365	116	1	leftPane	Awesome Cycles News	1200	FileModuleCachingProvider	NULL	NULL
5	0	0	56	55	366	116	1	footerLeftOuterPane	Products	1200	FileModuleCachingProvider	NULL	NULL
6	0	0	56	56	367	116	1	contentpaneLower	Header Images	1200	FileModuleCachingProvider	NULL	NULL
7	0	0	56	57	368	116	1	contentpane	Text/HTML	1200	FileModuleCachingProvider	NULL	NULL
8	0	0	56	59	370	116	1	footerLeftPane	Customer Support	1200	FileModuleCachingProvider	NULL	NULL
9	0	0	56	60	371	116	1	footerCenterPane	Company	1200	FileModuleCachingProvider	NULL	NULL
10	0	0	56	61	372	116	1	footerRightOuterPane	Connect	1200	FileModuleCachingProvider	NULL	NULL
11	0	0	57	62	373	116	1	footerRightPane	Contact Information	1200	FileModuleCachingProvider	NULL	NULL
12	0	0	57	63	374	116	1	footerLeftOuterPane	Products	1200	FileModuleCachingProvider	NULL	NULL
13	0	0	57	64	375	116	1	footerLeftPane	Customer Support	1200	FileModuleCachingProvider	NULL	NULL
14	0	0	57	65	376	116	1	footerCenterPane	Company	1200	FileModuleCachingProvider	NULL	NULL
15	0	0	57	66	377	116	1	footerRightOuterPane	Connect	1200	FileModuleCachingProvider	NULL	NULL
16	0	0	57	67	378	116	1	contentpane	About Us	1200	FileModuleCachingProvider	NULL	NULL
17	0	0	58	69	380	116	1	footerRightPane	Contact Us	1200	FileModuleCachingProvider	NULL	NULL
18	0	0	58	70	381	116	1	footerLeftOuterPane	Products	1200	FileModuleCachingProvider	NULL	NULL
19	0	0	58	71	382	116	1	ContentPane	Our Products	1200	FileModuleCachingProvider	NULL	NULL
20	0	0	58	72	383	116	1	footerLeftPane	Customer Support	1200	FileModuleCachingProvider	NULL	NULL
21	0	0	58	73	384	116	1	footerCenterPane	Company	1200	FileModuleCachingProvider	NULL	NULL
22	0	0	58	74	385	116	1	footerRightOuterPane	Connect	1200	FileModuleCachingProvider	NULL	NULL
23	0	0	59	75	386	116	1	footerRightPane	Contact Us	1200	FileModuleCachingProvider	NULL	NULL
24	0	0	59	76	387	116	1	footerLeftOuterPane	Products	1200	FileModuleCachingProvider	NULL	NULL
25	0	0	59	77	388	116	1	centerPane	Reach Us	1200	FileModuleCachingProvider	NULL	NULL
26	0	0	59	78	389	116	1	rightPane	Message Us	1200	FileModuleCachingProvider	NULL	NULL
27	0	0	59	79	390	116	1	footerLeftPane	Customer Support	1200	FileModuleCachingProvider	NULL	NULL
28	0	0	59	80	391	116	1	footerCenterPane	Company	1200	FileModuleCachingProvider	NULL	NULL
29	0	0	59	81	392	116	1	footerRightOuterPane	Connect	1200	FileModuleCachingProvider	NULL	NULL
30	0	0	59	82	393	116	1	leftPane	Visit Us	1200	FileModuleCachingProvider	NULL	NULL

Figure 11.37

If the module is not listed, you should look under Host Extensions [Your Module] Edit and make sure that Is Searchable is set to True. If it's False, then the manifest file of the module should be inspected. The module must have a line item for Searchable as a Supported Feature setting.

Figure 11.38 shows the manifest from HTML clearly indicating that there is a line-item for Searchable. Likewise, Figure 11.39 shows the extension settings of the HTML module and indicates that the value of Is Searchable is True.

```

dnn_HTML_dnn < X
</component>
<component type="Module">
  <desktopModule>
    <moduleName>DNN_HTML</moduleName>
    <foldername>HTML</foldername>
    <shareable>Supported</shareable>
    <businessControllerClass>DotNetNuke.Modules.Html.HtmlTextController, DotNetNuke.Modules.Html</businessControllerClass>
    <supportedFeatures>
      <supportedFeature type="Portable" />
      <supportedFeature type="Searchable" />
      <supportedFeature type="Upgradeable" />
    </supportedFeatures>
  </desktopModule>
</component>

```

Figure 11.38

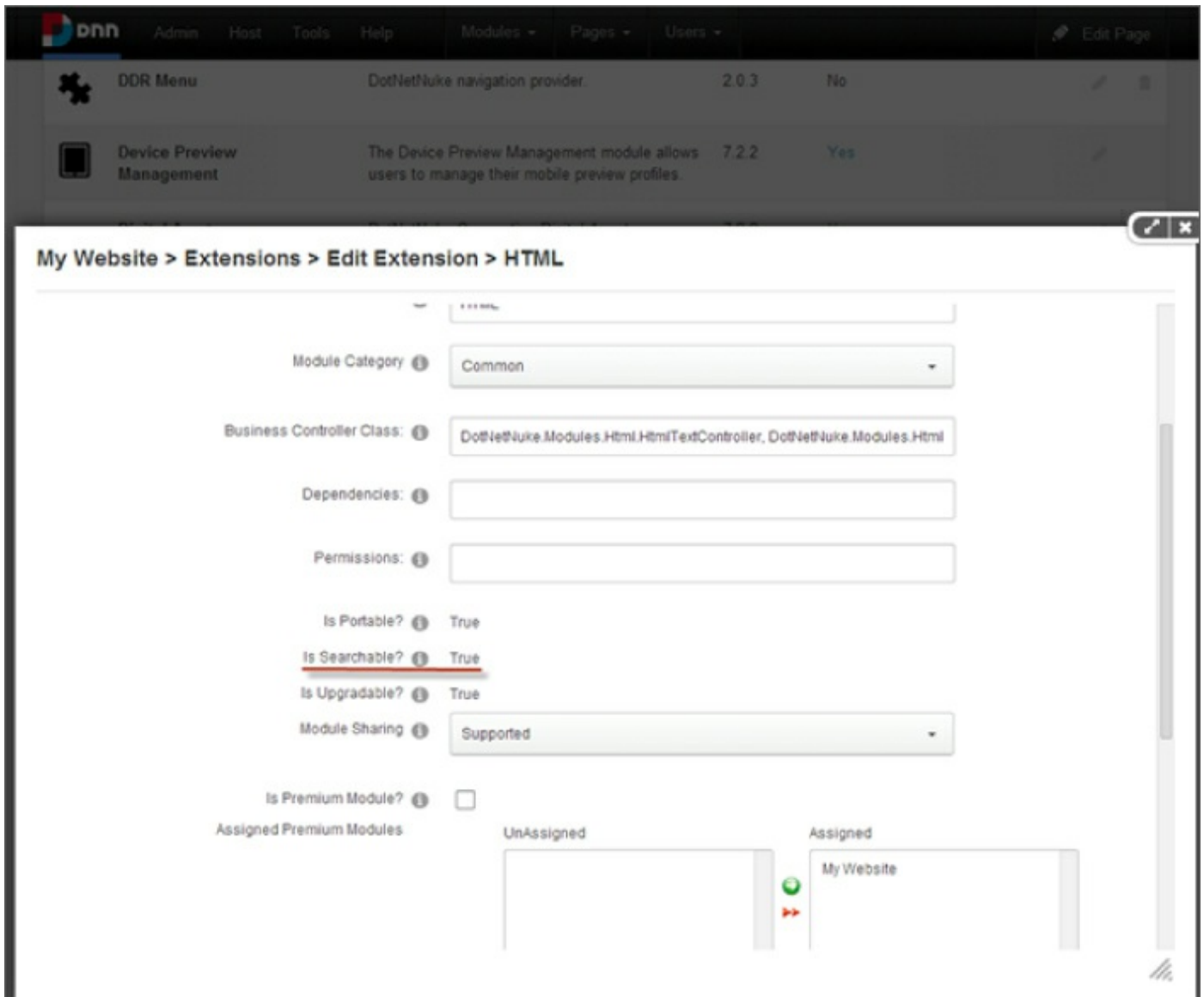


Figure 11.39

Enable Log4Net tracing by changing the level from `ERROR` to `ALL` (see [Figure 11.40](#)) in the `DotNetNuke.log4net.config` file at the root of the website folder. A change in this file may require an application pool recycle to see the effect.

```
DotNetNuke.log4net.config*  X
</layout>
</appender>
<root>
  <level value="ALL" />
  <appender-ref ref="RollingFile" />
</root>
</log4net>
```

Figure 11.40

In some cases Admin ↪ Search Admin ↪ Re-Index Content may need to be executed, after which the Host ↪ Schedule ↪ Search: Site Crawler ↪ Run may also need to be run. Once the Site Crawler is run, the most current log file under Portals_default\Logs folder will have trace information, as shown in the following code:

```
2014-01-17 14:13:46,488 [Alpha][Thread:35][TRACE]
DotNetNuke.Services.Search.
ModuleIndexer - ModuleIndexer: 1 search documents found for module
[DNN_HTML mid:379]
2014-01-17 14:13:46,526 [Alpha][Thread:35][TRACE]
DotNetNuke.Services.Search.
ModuleIndexer - ModuleIndexer: 1 search documents found for module
[DNN_HTML mid:398]
```

The previous code indicates that the DNN_HTML module was called to obtain `SearchDocuments` and the module returned one document. The HTML module is going to return 1 all the time as this module stores one instance of content for each instance of the module in the site. However, other modules such as Articles, Blogs, and so on, may return multiple documents.

Querying


There is a diagnostic tool available to find out how a query is analyzed by Lucene. By enabling tracing in Log4Net, you can easily see a lot of details. It is beyond the scope of this book to discuss the details of the following trace, but the key points to verify are that the Portal ID, Module Definition ID, and search type being passed and appear in the logs:

```
2014-01-17 14:39:58,521 [Alpha][Thread:36][TRACE]
DotNetNuke.Services.Search.Internals.LuceneControllerImpl -
Query: +(title:awesom title:awesome*) (tag:awesome tag:awesome*)
(description:awesom description:awesome*) (body:awesom body:awesome*)
(content:awesom content:awesome*) +(portal:[0 TO 0] portal:[-1 TO -
1])
+(moduledef:[116 TO 116] moduledef:[117 TO 117] moduledef:[116 TO
116]
moduledef:[117 TO 117] searchtype:[2 TO 2] searchtype:[3 TO 3])
+(locale:[1 TO 1] locale:[-1 TO -1])
-----
Awesome Cycles News
1.547232 = (MATCH) sum of:
  1.410534 = (MATCH) product of:
    2.350889 = (MATCH) sum of:
      1.81948 = (MATCH) product of:
        3.638961 = (MATCH) sum of:
```

3.638961 = (MATCH) weight(title:awesom in 26), product of:
0.3455227 = queryWeight(title:awesom), product of:
4.212703 = idf(docFreq=12, maxDocs=323)
0.08201924 = queryNorm
10.53176 = (MATCH)

Luke

The Lucene file can be analyzed outside of DNN. Luke is an open source, free Java tool that can be used to analyze the index. Luke can be downloaded from <http://code.google.com/p/luke/downloads/list> (see [Figure 11.41](#)).

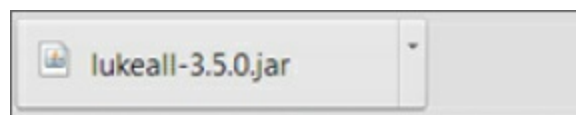


The screenshot shows a web browser window displaying the Luke download page. The page has a navigation bar with 'Project Home', 'Downloads', 'Wiki', 'Issues', and 'Source'. Below the navigation bar is a search bar. The main content area is a table of downloads. The table has columns for 'Filename', 'Summary + Labels', 'Uploaded', 'ReleaseDate', 'Size', and 'DownloadCount'. The table shows three rows of downloads: 'luke-src-4.0.0-ALPHA.tgz', 'lukeall-4.0.0-ALPHA.jar', and 'lukeall-3.5.0.jar'. The 'lukeall-3.5.0.jar' row is highlighted.

Filename	Summary + Labels	Uploaded	ReleaseDate	Size	DownloadCount
luke-src-4.0.0-ALPHA.tgz	Luke 4.0.0-ALPHA sources TGZ <i>Featured</i>	Jul 17	Jul 17	10.2 MB	1088
lukeall-4.0.0-ALPHA.jar	Luke 4.0.0-ALPHA standalone binary <i>Featured</i>	Jul 17	Jul 17	10.9 MB	6164
lukeall-3.5.0.jar	Luke 3.5.0 standalone binary	Dec 2011	Dec 2011	7.1 MB	28263

[Figure 11.41](#)

Given that Lucene is a file-based database, it is recommended that the jar file is downloaded and run locally on the server running DNN. Luke is started by simply double-clicking the downloaded jar file (see [Figure 11.42](#)).



[Figure 11.42](#)

Upon starting, Luke presents a Path to Index Directory dialog (see [Figure 11.43](#)). You should point to your website's App_Data\Search folder. It is highly recommended that the Open in Read-Only Mode checkbox be selected to prevent interference with the actual web site's search.

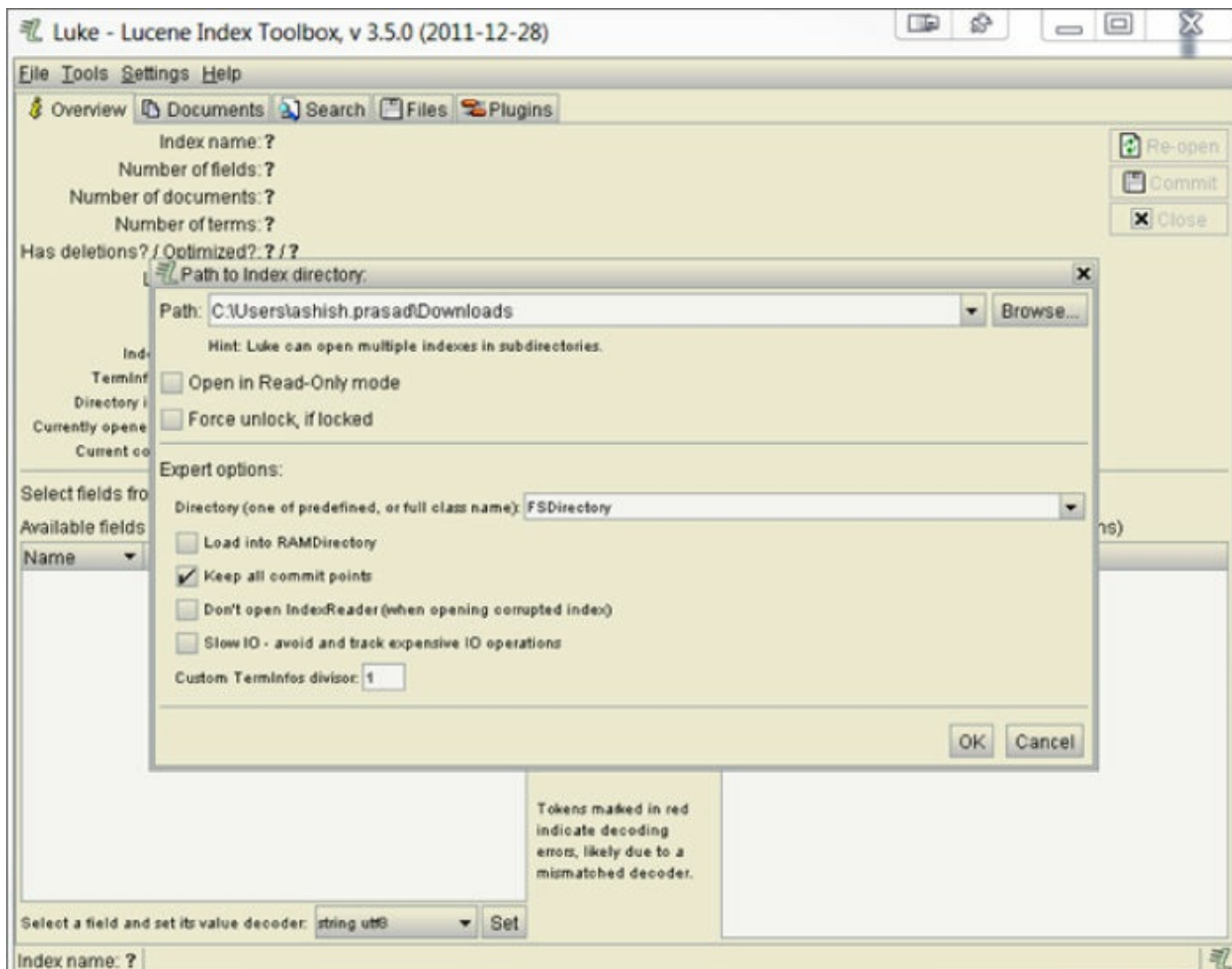


Figure 11.43

Luke provides many options (see [Figure 11.44](#)) for analyzing the index and performing custom querying. It is beyond the scope of this book to discuss those options. A quick search on the Internet should provide many how-to blogs and videos.

Luke - Lucene Index Toolbox, v 3.5.0 (2011-12-28)

File Tools Settings Help

Overview Documents Search Files Plugins

Index name: C:\source\DotNet\Nuke_CSWebsite\App_Data\Search

Number of fields: 16
 Number of documents: 182
 Number of terms: 1896

Has deletions? / Optimized?: No / No

Last modified: Sun Jun 16 16:47:21 PDT 2013

Index version: 36c2618b41a3
 Index format: 0 (Lucene 2.9)

Index functionality: lock-test, single norms, shared doc store, checksum, del count, omitT, user data, diagnostics

Terminfos index divisor: N/A

Directory implementation: org.apache.lucene.store.SimpleFSDirectory

Currently opened commit point segment: 5 (Sun Jun 16 16:47:21 PDT 2013)

Current commit user data: -

Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.

Available fields and term counts per field:

Name	Term count	%	Decoder
body	597	31.49 %	string utf8
content	45	2.37 %	string utf8
description	291	15.35 %	string utf8
key	182	9.8 %	string utf8
kw-keywords	1	0.05 %	string utf8
kw-meta	1	0.05 %	string utf8
kw-title	28	1.48 %	string utf8
locale	8	0.42 %	string utf8
module	72	3.8 %	string utf8
moduledef	8	0.42 %	string utf8
portal	16	0.84 %	string utf8
searchtype	10	0.53 %	string utf8
tab	71	3.74 %	string utf8
time	457	24.1 %	string utf8
title	105	5.54 %	string utf8
url	4	0.21 %	string utf8

Show top terms >>

Number of top terms: 50

Hint: use Shift Click to select ranges, or Ctrl Click to select multiple fields (or unselect all)

Terms marked in red indicate decoding error, likely due to a mismatched decoder.

Select a field and set its value decoder: string utf8 Set

Top ranking terms. (Right-click for more options)

No	Rank	Field	Text
1	182	locale	h
2	182	searchtype	l
3	182	time	X
4	182	searchtype	p
5	182	searchtype	d@
6	182	locale	t
7	182	searchtype	t
8	182	locale	x
9	182	time	P 0
10	182	time	T
11	182	locale	l
12	182	time	\
13	182	locale	'
14	182	locale	p
15	182	locale	l
16	182	searchtype	h
17	182	locale	d?
18	182	searchtype	x
19	182	searchtype	l
20	168	time	@ 0

Index name: C:\source\..._CSWebsite\App_Data\Search

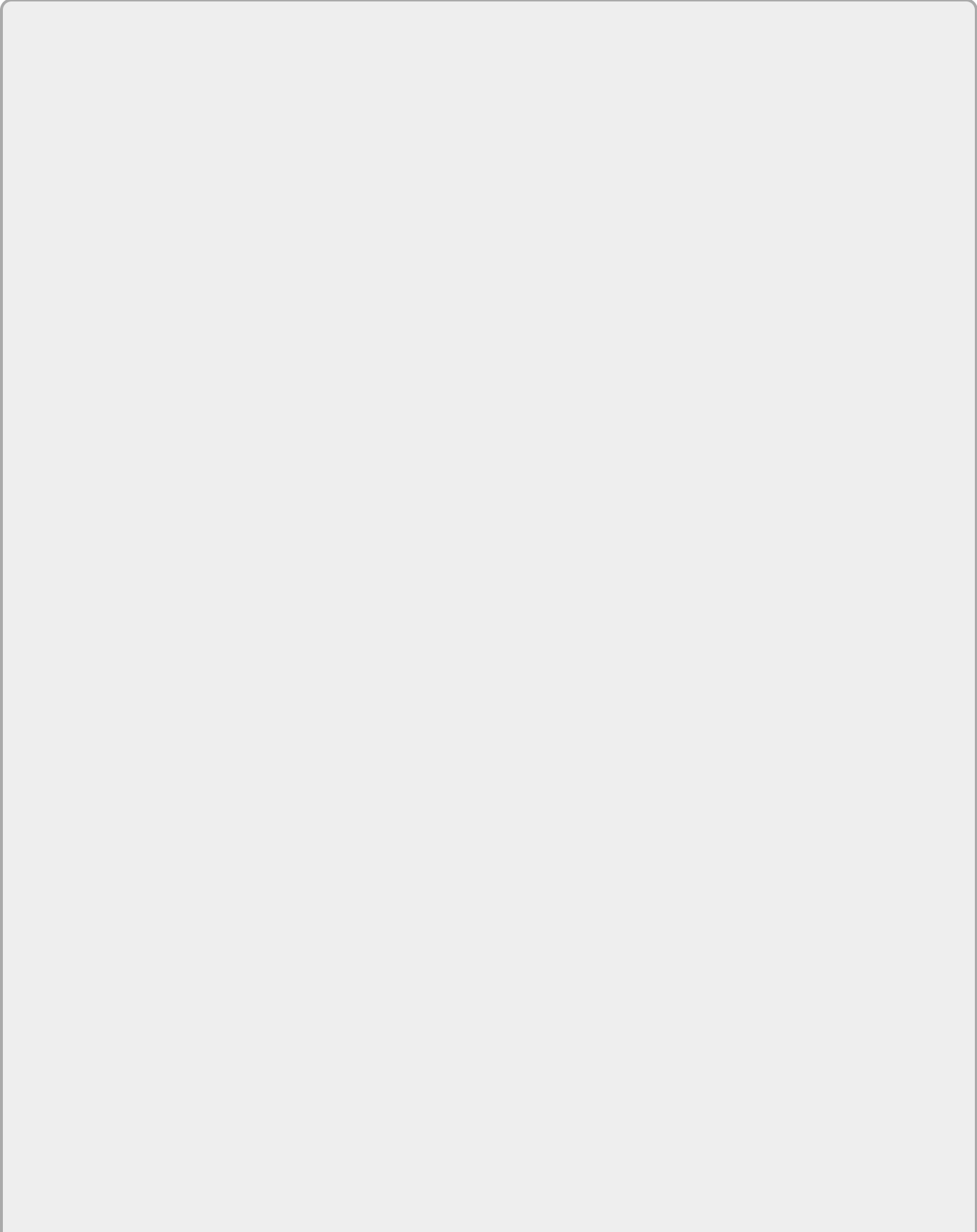
Figure 11.44

Summary

Search was written from scratch in DNN 7.1, which leverages Lucene.Net as its core indexing and querying engine. SQL Server is no longer used for search functionality. Search has a lot of new and enhanced functionality such as near real-time querying, speed, being locale-aware, improved security, ignore words functionality, synonyms, auto-preview, and more. Evoq products also have URL and File crawlers. Modules can integrate with the search feature by implementing the new abstract class `ModuleSearchBase`. A new crawler can easily be created by implementing `BaseResultController`.

Chapter 12

URL Management



What You Will Learn In This Chapter

- Creating URLs for DNN pages
- Configuring URLs for sites and installations
- Customizing URLs in Evoq
- Debugging URLs

Uniform resource locators (URLs) are vitally important to any website. Websites are essentially a series of URLs usually all sharing the same domain name. The more URLs there are in a site, the more complicated and important keeping track of those URLs becomes. This is particularly the case with a powerful and dynamic application like DNN, which can create thousands of URLs quickly and easily. URLs can be created for forum posts, blog posts, calendar entries, product catalogs—the list is endless.

This chapter walks you through the URL management tools and features within DNN and provides a guide for best-practice URL management for the developer and administrator of DNN sites.

The History of DNN URL Schemes

DNN has been through many versions from its inception, and throughout those versions the styles of standard URLs for DNN resources have changed. There are three main categories of DNN URLs: URLs for DNN pages, URLs for files and links, and URLs for other resources. This history of DNN URLs refers specifically to URLs for DNN pages.

TabId-Based URLs

If you look at the source code of a DNN installation, you will see that the standard application has only one ASP.NET page—`default.aspx`. This is the heart of the DNN application—all content is dynamically loaded by the `default.aspx` page, and every single different DNN page you can see is delivered by the `default.aspx` page.

The design has always been this way, and the early versions of DNN used a simple URL format that always referred to `/default.aspx`. The content that was displayed on the requested page was controlled by querystring key/value pairs. The simplest and most obvious of these was the Tab ID, which specified the DNN page (or tab, as they are known internally) that should be displayed. Thus, early DNN versions showed all their pages with a combination of `/default.aspx?TabId=XX`, where `XX` matched the `TabId` for a page within the `Tabs` table.

Friendly URLs Version 1

The `/default.aspx`-style URLs worked well and were relatively simple to parse and understand, but there was a problem. At the time, Google was going from strength to strength as the most popular search engine and was not reading and indexing querystring-based URLs. This meant that many sites built with DNN were not being ranked well in Google. To overcome this problem, **friendly** URLs were introduced into DNN. This change meant that the old `/default.aspx?TabId=XX` format was replaced by a new format that placed all of the page URL as part of the URL path. URLs in the new format were `/pagename/TabId/XX/default.aspx`. This worked around the problem of using querystring items to load the different DNN pages.

Any further items that are required to be in the querystring are also included in the URL path, so the loading of a specific module control creates a URL of `/pagename/TabId/XX/ctl/Settings/default.aspx`. This format can repeat

endlessly—any third-party extensions generating URLs would have their key/value pairs also included in the URL path.

This format could confuse people unfamiliar with the structure and reasons for it. Those used to a URL mapping directly to a physical path could not find the corresponding path on the actual server because all of the URLs were now virtual in that they didn't point to anything in particular. This was all achieved through the introduction of the Friendly URL Provider and URL Rewriter. The Friendly URL Provider is a plug-in provider that transforms a URL from the `/default.aspx?TabId=XX` format into the `/pagename/TabId/XX/default.aspx` format. The URL Rewriter is an HTTP module that reads all the requested URLs for a DNN application and **rewrites** them to a format that DNN expects—which is still the `/default.aspx?TabId=XX` format. These two additions to DNN add an extra layer of abstraction between the URLs that visitors see and the URLs that DNN code works with. The transformation of the URL between what the site visitor requests and the original DNN format was achieved by using regular expression matches and ASP.NET URL rewriting in the URL Rewriter.

Friendly URLs Version 2

While the new friendly URL format worked well in removing querystring items and also introducing the name of the DNN page into the URL so that the contents of the URL could be inferred, it still contained superfluous data that made the URLs confusing for people to read. Site visitors have developed an expectation that the URL of a page matches the contents, and search engines follow this and rank pages higher when the content matches the URL. In the pursuit of better reading for people and search engines, a second change was introduced to the Friendly URL Provider/URL Rewriter, which eliminated all mention of the TabId and `default.aspx` from the URL. With this change, a URL for a page became `/PageName.aspx`, where PageName was the name given by the site administrator for a page.

This simplified the URLs for a DNN page further, but only when the DNN page itself was in the URL. If extra information needed to be included in the URL, then the version 1 format discussed in the previous section was used as a fallback format.

This format is still largely in use for sites built with DNN and works relatively well. The first releases of DNN 7 continued to ship with the format, which became known as `humanFriendly` due to the configuration attribute used to

activate it.

Advanced URLs in DNN 7

With the release of DNN 7.1, a new URL Rewriter and Friendly URL Provider were introduced. Instead of replacing the existing URL Rewriter/Friendly URL Provider combination, the new functionality takes advantage of the flexible nature of DNN and is installed in parallel, allowing a choice of modes. The new functionality is called advanced URLs and is activated by default in all new DNN 7.1 and later installs.

There are many differences between the `humanFriendly` and advanced modes of URL rewriting/friendly URL generation. These are described in [Table 12.1](#).

Table 12.1 URL Mode Comparison

Behavior/Feature	human-Friendly mode
Home page URL	example.com/home.aspx
Ordinary DNN page URL	example.com/PageName.aspx
DNN page with third-party URL	example.com/pagename/TabId/89/ID/4440/default.aspx
404 handling	404 support only through ASP.NET native handling, which uses a 302 redirect to a generic error page
Enforcement of canonical URL for DNN page	None—the same page with duplicate URLs will resolve for all URLs
Support of page extensions	Only supports URLs ending in .aspx

Customization of DNN page URLs	Not supported—all URLs derived from the DNN page name
Enforcement of canonical domain for site	Choice of canonical link mode, redirect to canonical domain, or no action
Support for URL customization for third-party extensions	Not supported
Tuning of URL behavior through regular expression filters	Supports only URL rewrites and redirects

There are a lot of other smaller features of the advanced mode that aren't listed in [Table 12.1](#). These will be mentioned in the chapter where necessary.

DNN 7.1 and 7.2 will not swap the mode from `humanFriendly` to `advanced` during a DNN upgrade. Owners of existing DNN sites will have to manually change the URL mode of their site by altering the configuration file after an upgrade to 7.1 or later version.

Switching an Upgraded Site to Advanced Mode

If you have upgraded a site to DNN 7 and have not activated the advanced mode, the following steps will guide you through the simple process:

1. Open the `web.config` file for your DNN site with a text editor.
2. Do a search for the term “FriendlyUrlProvider.” You should find the following line in the `web.config` file:

```
<add name="DNNFriendlyUrl"
type="DotNetNuke.Services.Url.FriendlyUrl.DNNFriendlyUrlProvider,
DotNetNuke.HttpModules" includePageName="true" regexMatch="^[a-zA-
Z0-9 _-]"
urlFormat="humanFriendly"/>
```

3. Modify the `urlFormat` attribute to match the following:

```
<add name="DNNFriendlyUrl"
type="DotNetNuke.Services.Url.FriendlyUrl.DNNFriendlyUrlProvider,
DotNetNuke.HttpModules" includePageName="true" regexMatch="^[a-zA-
Z0-9 _-]"
urlFormat="advanced"/>
```

4. Save and replace the `web.config` file back into the root folder of your DNN Application.

This process changes the mode of the URL rewriting and friendly URL generation within your DNN application and will change the way the URLs look and work on your site. Generally this is a low-risk process, as the two modes work in similar ways, but it is advisable to test this thoroughly to ensure that all site functions work as expected. It is good practice to set up a test version of the site to make this change before applying it to a live website. Testing should pay particular attention to the operation of any third-party extensions and key functions like file uploads, image generation, site editing, and so on.

SEO Focus on URLs

A lot of the functionality in advanced mode is focused on search engine optimization (SEO). SEO is a broad term and encompasses all of the actions taken to work toward higher ranking of sites in search engines. Although an in-depth discussion of SEO is beyond the scope of this book, you may be interested in how URLs can affect this important topic. Dealing with URLs in particular, the SEO focus is around three main points.

- Elimination of duplicate URLs, where two different URLs will load the same page of content. This causes a page to be ranked lower than it would otherwise be, as search engines don't know which is the correct URL to list.
- Simplification of URLs to remove superfluous ID and other technical information from the URL and to reduce it down to describing the purpose of the page as succinctly as possible.
- Redirecting requests to existing or legacy URLs to new locations when a site changes. This preserves the value of any external links pointing at the site so that visitors and search engines can use existing links and not end up on a 404 page.

The remainder of this chapter focuses on the advanced mode features that are new for DNN 7. The scope of the discussion will stay within the features available within the DNN Platform. Any discussion of features available only in the commercial Evoq solutions will be clearly delineated.

Understanding URL Structure in DNN

DNN URLs can be virtual URLs, which load content into a page; static resources such as files, images, and scripts; as well as API services and custom code created in extensions. Understanding each of these URL types is important in understanding how DNN works, how to extend it, and how to troubleshoot problems when requests are not working as expected. [Table 12.2](#) provides example URLs that DNN sites will use.

Table 12.2 DNN URL Types

Type	Example	Use
User profile URL	example.com/bill-smith	Points to specific user profile page for the user identified by his or her profile URL.
Page URL	example.com/page-name	For loading DNN page and all default view modules that page.
Page URL with module	example.com/page-name/ctl/Edit/mid/876	Loads the Edit content for the module via a Module of 876 and other controls.
Page URL with third-party	example.com/page-name/cid/345/content-name	For loading DNN page and all default view modules.

content		that page and speci content c module relating t the cid paramete
Service call	example.com/DesktopModules/API/ModuleName/MethodName	Calls the MethodNa: procedur the Servi API for th ModuleNa:
Site-specific resource	example.com/portals/0/image.jpg	Static ima: file locate the site-specific folder for site identified a Portal I O.
Sitemap handler	example.com/sitemap.aspx	Returns t search engine sitemap f the speci: portal. Th maps to a entry in t <handler section o web.conf file.
Keep alive	example.com/keepalive.aspx	Points to specific k alive file the serve

		which can be pinged by an external service to stop the installation from being installed by the IIS server.
Module-specific resource	example.com/DesktopModules/ModuleName/module.css	Module-specific CSS file, located in the ModuleName folder.

There are many more URL types that can be seen during the loading of a DNN page. Developers who are extending DNN will generally be creating new URLs for loading specific module controls, displaying different content on a page, building new Service API calls, and loading site-specific resources.

The important concept to learn from this table is that many DNN URLs are virtual and do not point to physical resources on the server. They are transformed by the DNN framework either through the URL rewriter or through the services layer into a form that the underlying web server understands. Developers must learn these patterns so that they can create and leverage the correct URLs for the intended use. The following section details how to correctly generate and manipulate URLs in DNN extensions.

URL Configuration and Customization

In this section, you learn the correct ways to create URLs for different purposes when creating extensions for DNN. As shown in [Table 12.2](#), DNN URLs have a specific pattern for specific purposes, and it is important to adhere to these patterns to ensure compatibility with other extensions and a consistent upgrade path for future DNN versions.

Creating Custom URLs for DNN Pages

DNN pages are identified using the name of the page or a specific URL created for that page. The URL for a DNN page can be updated through the Page Settings pop-up for any DNN page. In a default DNN installation, all URLs for DNN pages are created by deriving a URL from the page name.

It is possible to change the URL of a page by changing the name of the page, but in many cases it is desirable for the name of the page and the URL of a page to have different values.

To modify the URL for a DNN page, follow these steps:

1. Ensure you are logged into the DNN site as either a host or admin-level user.
2. Navigate to the page you want to modify the URL for, hover your mouse over the Edit Page drop-down, and click Page Settings.
3. Find the Page URL field, and type the URL that you want to use for the specific page. Ensure that you use only URL-legal characters. Any characters that are not URL-legal will be removed from the URL.
4. Click Update Page to save the changes.

The page will redirect to the new URL, which now displays in the browser address bar.

[Figure 12.1](#) shows the Page Settings dialog where a custom URL value can be entered.

The screenshot shows the 'Advanced Settings' tab for a page in DNN. The 'Page Name' is 'About Us', the 'Page Title' is 'About Our Company', and the 'Page URL' is 'en.dnn7.local /About-Awesome-Cycles'. The 'Do Not Redirect' checkbox is unchecked. The 'Description' is 'Awesome Cycles is a fantastic personal mobility company who is committed to providing our customers the very best in person-powered wheeled transport.' The 'Keywords' field is empty.

[Figure 12.1](#)

Once a URL is changed, a system redirect is also created so that any references to the old URL are redirected to the new URL. This is done automatically. The same process applies to renaming the page (and indirectly changing the URL)—when a URL is changed by renaming the page, a redirect is created to route requests for the URL derived from the old page name to the URL derived by the new page name.

Advanced URL Creation for Evoq

The Evoq editions of DNN contain the ability to create further customization for URLs. The screens shown in this section are not available in the base DNN Platform.

The URL Management section in Evoq can be found on the Page Settings pop-up, in the Advanced Settings tab. This allows for the creation of custom URLs and URL redirects for the specific page. The URL Management section contains two lists: Custom URLs and System Generated URLs. Custom URLs are those that are specifically created by an administrator. System-generated URLs are those automatically created by DNN to redirect old DNN URL versions and to redirect any prior DNN URLs that have been used with the page in the past. Renaming the page or creating a new custom URL will cause the prior URL to appear in the list of system-generated URLs. This list is not exhaustive—there will always be some combinations of prior URLs that do redirect to the page but do not appear in the list of system-generated URLs.

This is not a problem or bug but reflects a need to preserve UI space and avoid confusing first-time users with long lists of esoteric URLs.



NOTE

Advanced URL settings and management options for Evoq are discussed in the “Site- and Installation-Level URL Configuration” section later in the chapter.

Creating a Custom URL in Evoq

The custom URL functionality shown in the previous section is available in Evoq versions. This is limited in the options that can be used, while the Custom URLs section allows for more complex URL requirements. Custom URLs created here can have the following properties:

- The URL can be created with a site alias other than the primary site alias. This can also be used for child pages to inherit.
- For multi-language sites, different URLs can be created for different languages.

Follow these steps to create a custom URL in Evoq:

1. Ensure you are logged into Evoq using a host or admin user.
2. Navigate to the page you want to create a custom URL for.
3. Open the Page Settings by hovering the mouse over the Edit Page menu and clicking Page Settings.
4. In the Page Settings pop-up, select the Advanced Settings tab.
5. Scroll down the pop-up and expand the URL Management section.
6. Click the Create button to show the URL input fields.
7. Enter the field values as required. See [Table 12.3](#) for a detailed explanation of the fields.
8. Click Save when you finish.
9. Scroll to the bottom of the pop-up, click Update Page to close the settings, and save the changes.

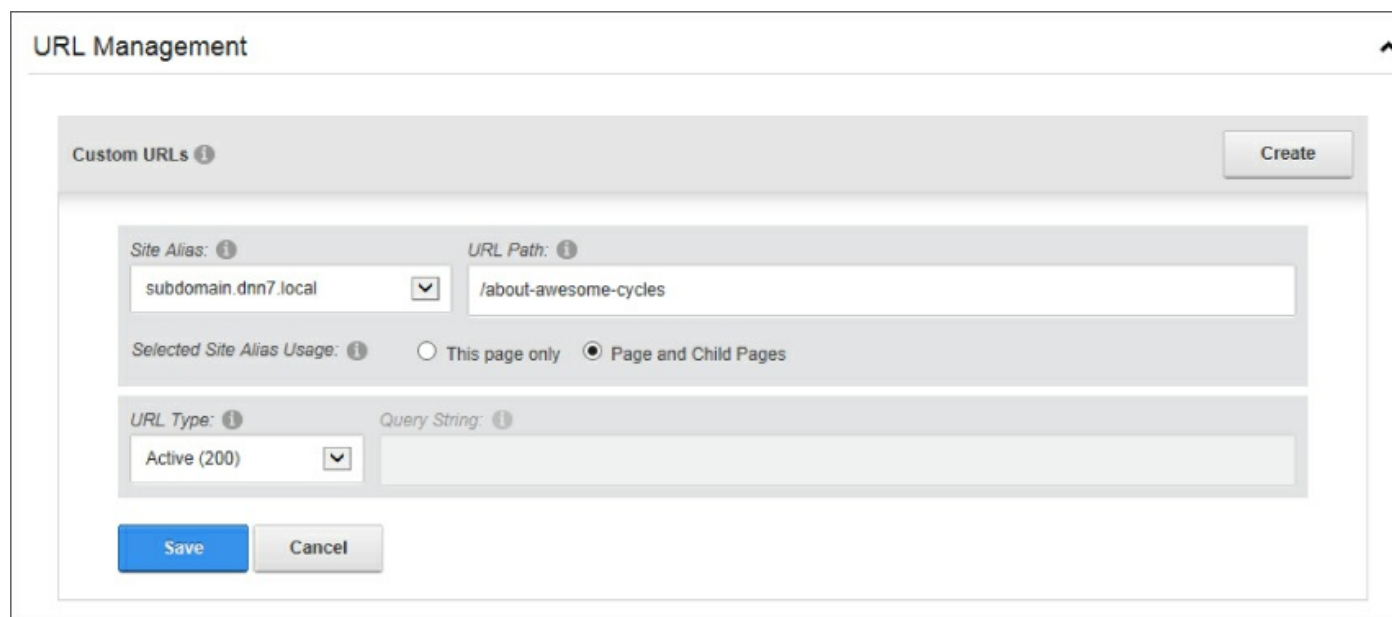
Table 12.3 Create URL Fields

Field	Description
-------	-------------

Site Alias	The list of site aliases in the current site. By default, the primary alias will always be selected. Change this to a different alias if you want the custom URL or URL redirect to be associated with a different alias.
URL Path	The URL to redirect. This URL path (along with the site alias) will be matched and associated with the page. The path automatically does partial matches— example.com/new-path will match, as will example.com/new-path/something-else . The leading/character is automatically added to the URL path. It is possible to create an empty path (leaving only the site alias), which can associate a top-level domain name with a specific DNN page that is not the home page. However, care must be taken not to create a duplicate URL of the site home page.
Selected Site Alias Usage This Page Only or Page and Child Pages	When a site alias other than the primary site alias is chosen, the option on how to use that alias appears. <i>This Page Only</i> associates the selected site alias only with the current DNN page. <i>Page and Child Pages</i> associates the current page and any and all child pages with the selected site alias. This option means that any pages created and using the current page as a parent page will also use the selected site alias without having to create custom URLs for each child page.
URL Type Active (200) or Redirect (301)	The URL Type option changes behavior between becoming the active URL for the page (returning a 200 HTTP status code) or issuing a permanent redirect to the DNN page (returning a 301 HTTP status code). When the Active (200) option is chosen, the URL becomes the standard generated URL for the DNN page, and each menu link will use the custom URL.
Query String (Only active when URL Type = Redirect)	The Query String field allows the association of a particular URL querystring with a custom URL redirect. This allows the redirect of previously existing URLs that contain a querystring. An example is the redirection of a legacy URL of example.com/index.aspx?page=about to a new About page—in this case <code>?page=about</code> would be entered as the Query String value.
Language	The Language drop-down allows the association of a specific

(Only active when multiple languages are installed) language with a custom URL. This allows the creation of language-specific URLs for a page. When this option is used and the URL is requested, the specific language code is added to the rewritten URL. This option is often used to create ASCII character versions of different language values for simpler URLs in a website.

The Custom URLs section showing the creation of a custom URL looks like [Figure 12.2](#).



The screenshot shows the 'URL Management' interface. At the top, there is a 'Custom URLs' section with a 'Create' button. Below this, there are two main input areas. The first area is for 'Site Alias' and 'URL Path'. The 'Site Alias' dropdown is set to 'subdomain.dnn7.local' and the 'URL Path' text box contains '/about-awesome-cycles'. Below these fields, there are radio buttons for 'Selected Site Alias Usage', with 'Page and Child Pages' selected. The second area is for 'URL Type' and 'Query String'. The 'URL Type' dropdown is set to 'Active (200)' and the 'Query String' text box is empty. At the bottom of the form, there are 'Save' and 'Cancel' buttons.

[Figure 12.2](#)

Creating a Custom URL Redirect in Evoq

Creating URL redirects is important when reorganizing sites or replacing old sites with new sites. The same URL creation fields can be used to create a URL that will permanently redirect to the page for which it is created. When a redirect is created in this way, any URL path after the segment that identifies the DNN page is carried over as the path for the new page. This allows items such as blog posts, forum threads, and other path-based content to work as before when the DNN page changes.

Follow these steps to create a URL redirect with Evoq:

1. Ensure you are logged into Evoq using a host or admin user.
2. Navigate to the page for which you want to create a custom URL.

3. Open the Page Settings by hovering the mouse over the Edit Page menu and clicking Page Settings.
4. In the Page Settings pop-up, select the Advanced Settings tab.
5. Scroll down the pop-up and expand the URL Management section.
6. Click the Create button to show the URL input fields.
7. Enter the field values as required. See [Table 12.3](#) for a detailed explanation of the fields. The URL to redirect will go in the URL Path field.
8. Ensure that the URL Type field is set to Redirect (301) and click Save.
9. Scroll to the bottom of the pop-up and click Update Page.


It is possible to enter many redirects for a single page, allowing many different URLs to all be redirected to the DNN page. It is not necessary to create redirects for prior versions of DNN URLs for a page, unless that page (and the underlying tab record) no longer exists.

[Table 12.3](#) describes the fields in the Create URL section.

Testing Custom URLs and Redirects in Evoq

Evoq solutions include a testing facility where testing of new custom URLs and URL redirects can be performed. Testing of URL changes is vital to verify that the URL works as expected but also to assist in understanding how the URL is being generated and rewritten when DNN pages are being requested.

The test URL functionality tests the generation of URLs from within DNN, as well as testing rewriting and redirecting of the URLs. Understanding this behavior is essential to developing a full understanding of the way in which DNN ultimately works, and the test URL functionality helps to reveal what goes on when pages are being generated and requested.

To test a DNN URL, go to the Admin  Advanced URL Management page, and click the Test URL tab. The Test URL section is separated into two halves, the first for testing generation of the URLs and the second for testing rewriting behavior. The two are related but independent and can be used separately.

Follow these steps to test the generation of a URL:

1. Select the page you want to test, either by locating in the list or by using the Search functionality to find the page required. The pages are in alphabetical order rather than the display order of the site.

2. If the URL you want to test contains extra parameters apart from just the page name, enter these as a querystring in the Query String box. This is optional—there is no requirement to use this box. Entering values as the Query String value acts in the same way as providing parameters in the `NavigateURL` API call when developing custom DNN code. The format of the Query String value should be `&key=value&key2=value2`. You can mimic the creation of DNN control parameters such as `&ctl=edit&mid=345`.
3. If the URL you are testing would normally utilize a different `pageName` input when the URL is generated, enter this value in the Custom Page Name/URL End String box. This generally applies to URLs generated for some third-party extensions, which sometimes supply a “friendly” end to the URL. This is quite common in blog and article modules. In this case, you would enter the value directly as it is passed into the `NavigateURL` API call. An example for a blog post might be `my-test-blog-post.aspx`.
4. When all of the inputs are correct, click the Test URL button.
5. The generated URLs will now appear in the Resulting URLs section. A URL will be generated for each site alias/site language combination for the current DNN site.

While it may be confusing to see multiple URLs generated for a single page, what this means is that the generation of URLs from within the DNN Platform is dependent on multiple inputs—different inputs naturally give different outputs. Rather than provide a bewildering array of inputs to provide a single output, the list of inputs is restricted and multiple results are generated. When viewing the results, it is important to understand which is the primary alias for the specific site/page combination and as such which of the generated URLs would be used for display in different cases, such as the DNN menu or sitemap, and in internal content use such as a list of forum threads, blog posts, or a product catalog.

Each of the generated test URLs is a clickable hyperlink; these do not go to the actual page, but instead load that URL into the input box for the test URL rewriting function.

[Figure 12.3](#) shows the test URL generation function, with a Query String input of `&key=value` and a URL ending value of `other-url-ending.aspx`.

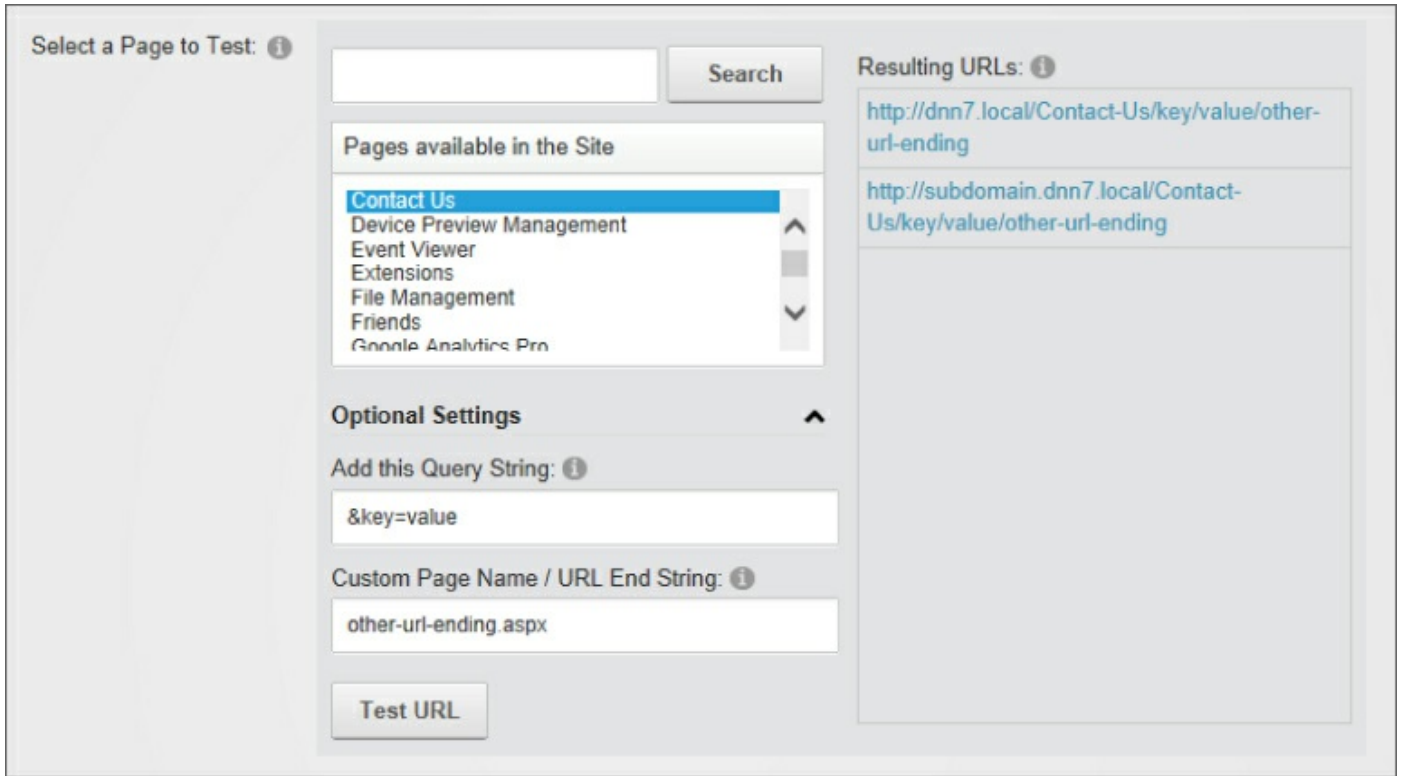


Figure 12.3

Follow these steps to test a URL for rewriting:

1. Enter a value in the test URL rewriting input box or click a test URL generation result hyperlink to copy the URL. The URL must be fully qualified including the http:// (or https://, if applicable).
2. Click the Test URL Rewriting button to submit the URL to the rewriting engine.
3. Find the results of the test under the test URL rewriting input box and button, and interpret the test results using [Table 12.4](#).

Table 12.4 Rewriting Test Result Fields

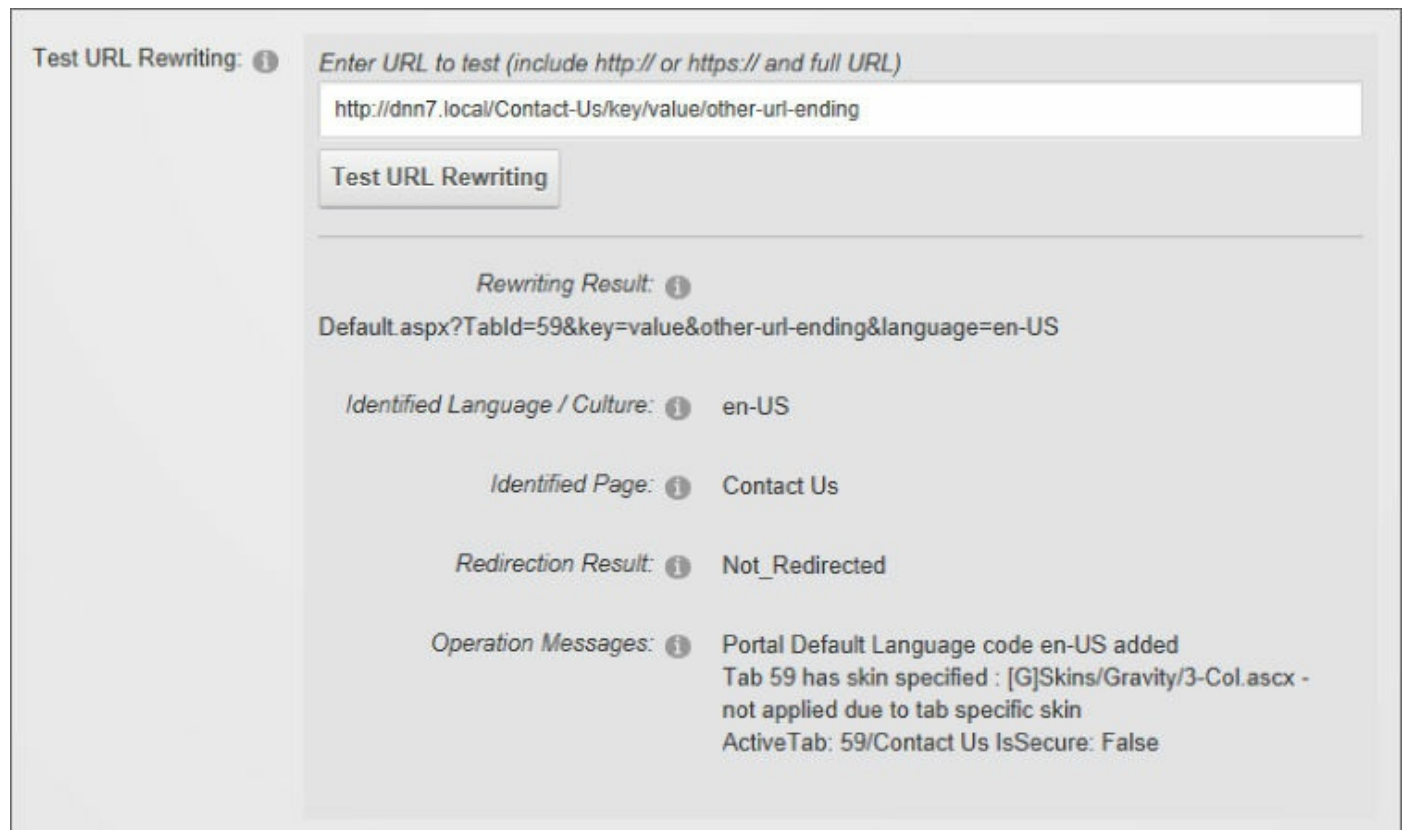
Output Field	Expected Results	Description
Rewriting Result	A rewritten URL showing the /Default.aspx path with a Query String value. A simple result will be Default.aspx?	This is the raw, rewritten URL that DNN code will use for controlling page output. The rewritten URL will usually (but not always) contain a TabId but also may specify a PortalId to specify the site and any third-party site contents.

	TabID=87.	If a querystring was input into the Test URL Generation section and the result copied into the test URL rewriting input box, the querystring should be replicated in the rewriting output. This shows how a querystring is converted into a friendly URL and then back into a querystring by the rewriting process.
Identified Language / Culture	A language/culture code in the form of xx-YY where xx is the language identifier and YY is the culture identifier. Examples are en-US, fr-FR, and so on.	Every DNN site has a language associated with it, and many DNN sites have more than one language associated. The language/culture code is either explicit in the URL or implied through a combination of settings, including the default language for the site. This output field shows explicitly which language would be loaded and used for the requested URL.
Identified Page	The name of the DNN page identified by the request. The output will show either the name of the page or None if no specific page was identified from the URL.	The majority of DNN URLs identify a specific page in the request, and those without a specific page may load the home page by default. The Identified Page field shows which page has been identified by the URL and which would be loaded as the ActiveTab within DNN. This controls which page is shown by the URL.
Redirection Result	An internal action code used by the URL rewriting process. Exposing this action in testing helps understand how/if the URL will be redirected.	The full list of actions is contained within the <code>RedirectReason</code> type found in DNN code. The value depicts why a URL will or will not be redirected. This code is important to developers who are using custom code to control the action of URLs.

<p>Operation Messages</p>	<p>A list of coded messages that reveal the logic path taken for the URL rewriting process.</p>	<p>The list of messages changes with each request and is dependent on the URL, the configured options, and any extension URL providers installed.</p> <p>URL messages will often include a description of which Tab ID was found, whether it was a secure page (SSL is on for the page), and whether the page loads the site-default skin or has a page-specific skin. Extension URL provider developers can add their own custom messages to this list to help with debugging.</p>
---------------------------	---	---

Understanding the rewriting results requires a good knowledge of how DNN works but is also a useful tool for learning the internals of DNN. [Table 12.4](#) shows the output fields, expected results, and meaning of what the results say.

[Figure 12.4](#) shows the result of testing the URL rewriting for the results of the URL generation test.



[Figure 12.4](#)

The testing section should be used both by administrators and DNN developers as a quick and easy way to test that custom URLs and URL redirects are working as expected. It is very difficult to verify redirect results using a browser, as the operation is opaque to the viewer. The test utility allows for detailed testing without using any other specialized tools.



Site- and Installation-Level URL Configuration

Multiple configuration options control the appearance and behavior of DNN URLs. Many of these options are duplicated at the site and installation levels. This allows different sites within overall DNN installation to appear and behave in a different way.

Installation-Level Configuration Options

[Table 12.5](#) shows the configuration options and their location within the DNN administration menus.

Table 12.5 Installation-Level Configuration Options

Page	Tab	Section	Description
Admin  Site Settings	Advanced Settings (all versions)	Site Aliases	Controls settings over the site aliases used
	Advanced Settings (all versions)	SSL Settings	Controls what the SSL and standard domains for the site are
	Advanced URL Settings (Evoq only)	URL Settings	Controls URL redirect and DNN page URL generation settings
	Advanced URL Settings (all versions)	Extension URL Providers	Configuration of installed extension URL providers
Admin 	General	Configuration of options that control the	

Advanced Url Management (Evoq only)		behavior of URL generation and redirection
	Regular Expressions	Configuration of regular expressions that control behaviors of URL rewriting, redirection, and generation
	Test URL	Tools to test URL generation and URL rewriting in the site

Site aliases are an important area to understand when configuring a DNN site. An alias is the name given to the unique combination of the top-level domain name and URL path that prefixes all DNN page URLs for a specific site. In many cases this is just a domain name, but the alias is often a domain name plus path either to create a child site or to specify a language path for a site. [Table 12.6](#) shows some example site aliases.

Table 12.6 Example Site Aliases

Example Site Alias	Type	Description
www.example.com	Normal top-level domain for a site	The most common type of alias in DNN websites
www.example.com/child	Alias for a child site	Used to create a new site within a DNN installation, where another site exists using the top-level domain. A child path allows a completely separate DNN site to be created while sharing the same domain
www.example.com/en www.example.com/nl	Aliases for a site that specify a language	Used to associate a different language with a request, depending on which alias is used within a URL

Site Alias Configuration

The Site Aliases section contains options for controlling the behavior for site-level actions that can be associated with specific aliases. [Figure 12.5](#) shows

the Site Aliases section.

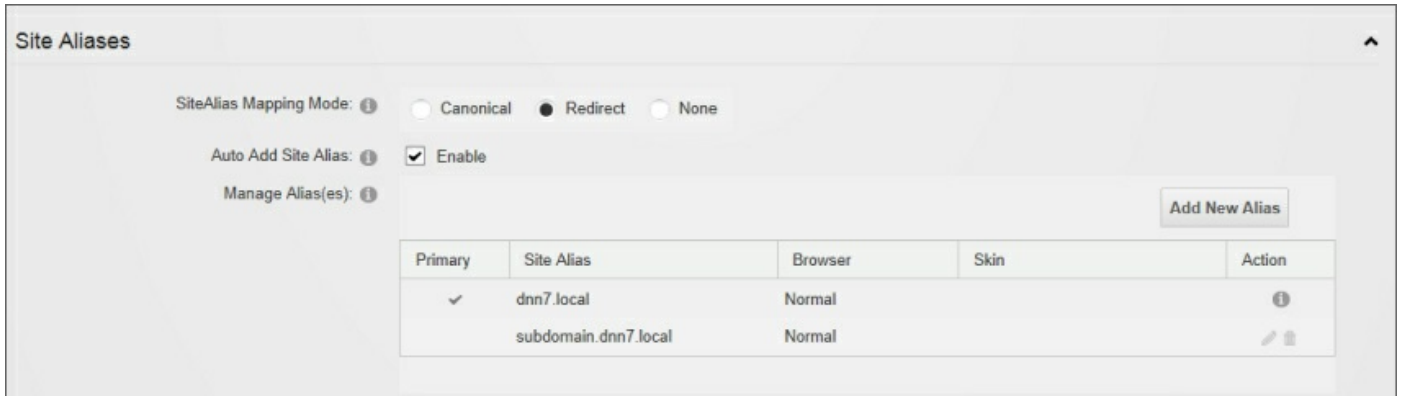


Figure 12.5

The individual options in this section are shown in [Table 12.7](#).

Table 12.7 Site Alias Configuration

Configuration Option	Description
Site Alias Mapping Mode Options: Canonical Redirect None	<p>The mapping mode controls the behavior of the aliases when there is more than one alias in the list.</p> <p>Canonical allows any of the aliases to be used, but will emit a canonical link element in the page headers using the primary site alias as the root of the canonical URL.</p> <p>Redirect means that any request to a site alias that is not the primary site alias will be redirected (using a 301 Permanent Redirect) back to a URL using the primary alias as the root of the URL.</p> <p>None allows the site to be requested with any alias in the list. All other URLs then seen on the site will use the requested alias.</p>
Auto Add Site Alias Options: Checked/Unchecked	<p>The Auto Add function switches on a behavior where any new, valid site alias requested will be added to the list of site aliases. In practice, this means that any domain name bound to the underlying IIS website can be used to request the website.</p>
Manage Aliases	<p>The Manage Aliases table lists the current site aliases configured for the site. Within this list are multiple columns that can change depending on other options</p>

for the site.

The Add New Alias button is used to add a new site alias to the site. For a site alias to work correctly, it must also be bound to the underlying IIS website.

[Table 12.8](#) contains a breakdown of the individual site alias values, as shown in the site alias grid.

Table 12.8 Site Alias Values

Field	Purpose
Primary	Used to mark an alias as the primary alias for the site. The primary alias controls behavior for the Redirect and Canonical Mapping modes. Note that there can be greater than one primary portal alias when different languages or browser types are used.
Site Alias	The actual domain name and (optionally) URL path used to reach a DNN site. The site alias never includes a leading scheme/qualifier, such as http or https, and never includes a trailing /.
Language	When a site has multiple languages installed, the Language field allows one of the installed languages to be associated with a specific alias. This allows use of example.com/xx-YY to specify a language, or some other combination, such as xx-YY.example.com . The Language field is visible only if the site has more than one language installed. In single-language DNN sites, it will not be visible.
Browser	Associates a site alias with either normal or mobile sites. This is used when a particular alias is to be associated with a specific browser type, such as using m.example.com to serve mobile content.
Skin	Links a specific skin installed in the site with a specific alias. When the Skin value is set, that skin is loaded by default when the site is requested with the site alias.

SSL URL Configuration

The SSL Settings section controls when HTTPS URLs are used for DNN sites and which URLs are to be used. The SSL settings work in conjunction with

the Is Secure property on each DNN page and control the appearance of the URL used for SSL/non-SSL pages.

[Table 12.9](#) shows the options and purpose of the SSL settings.

Table 12.9 SSL Settings

Field Name	Values	Description
SSL Enabled	Checked/unchecked	This check box provides a site-level switch for enabling/disabling SSL URLs.
SSL Enforced	Checked/unchecked	When checked, SSL or non-SSL (HTTP/HTTPS) URLs are enforced by issuing a redirect to the SSL URL.
SSL URL	Valid site alias	An optional field. If primary site alias does not have a valid SSL certificate, an alternate site alias with an SSL certificate can be supplied. This site alias will then be used for the DNN pages marked as requiring SSL.
Standard URL	Valid site alias	An optional field—if an SSL URL is supplied, the standard site alias to use for non-SSL pages must be listed here. This is generally the same as the primary site alias.

Evoq Advanced URL Settings

In Evoq editions, the Advanced URL Settings tab in the Site Settings page contains the basic options used for controlling URL behavior. In the base DNN Platform it contains the listing and configuration options for extension URL providers.

[Table 12.10](#) shows the Evoq edition configuration options.

Table 12.10 Evoq Advanced URL Settings

Field	Values	Description
Redirect “old” URLs to new Friendly URLs	Checked/unchecked	This field enables functionality to automatically redirect any URLs that are not the canonical URL. This is generally used to ensure that no URLs of the tab

		ID or earlier friendly formats are used for the site. Disabling this option does not affect the functionality of custom URL redirects created. The default and recommended value is for this field to be checked.
Convert URL to lowercase	Checked/unchecked	The lowercase conversion option converts all URLs generated by the site to all lowercase. Administrators often use this site both for aesthetic reasons and to ensure that no search engines regard lowercase and mixed-case versions of the same URL to be considered duplicates of each other.
Page URL Concatenation	Checked/unchecked with options for - and _.	DNN page URLs are generated based on the DNN page name. If the page name is two or more words, the spaces are removed to create a valid URL. This option allows the replacement of those spaces with either a hyphen (-) or an underscore (_). If disabled, the spaces will be removed and the words are concatenated to create the URL. The replacement character also works on URL illegal or URL reserved characters that appear in DNN page names.
Page Extension Handling	Checked/unchecked	The extension refers to the file extension on the end of URLs. Traditionally this was a way for IIS to associate different applications with different page extensions, and traditional ASP.NET requests always ended with .aspx. With .NET version 4, this is no longer a requirement, and thus DNN 7 has an option to remove the .aspx extension.
Handle	301 redirect to	DNN allows pages to be deleted, to have

deleted, expired, and disabled pages by	home page <i>or</i> 404 error page shown	expiration dates applied. When a URL is requested that points to a deleted, expired, or disabled page, this option controls what the behavior should be. Regardless of this setting, it is possible to create a redirect for a deleted, disabled, or expired page by creating a custom redirect. Additionally, this behavior does not apply to pages that are hard deleted and no longer exist in the site Recycle Bin.
---	--	---

Extension URL providers are explained in the “Custom Extension URL Providers” section in this chapter.

Evoq Advanced URL Management Options

The Evoq editions of DNN include an extra page that contains more configuration options for URL management. It is important to note that these options are still present in the underlying DNN Platform (where they are implemented at a code level), but the DNN Platform contains no UI to control the behavior. For the DNN Platform, the configurations are set to default upon install and can be changed only using third-party tools or by directly modifying the database values.

Some of the configuration options shown are duplicated in the Settings page and the Advanced URL Management page. This is to show the simple options to administrators and allow them to ignore the more complicated values.

The Advanced URL Management page is visible in both the Admin and Host menu groups. This refers to two separate pages; however, the values on the pages are largely the same. [Table 12.11](#) refers to host-only specifications where necessary; otherwise, the values are in both pages. The difference in the two pages is that the host-level configurations apply to the entire installation, while the Admin page applies only to the current site the page is loaded for. This creates some conflicts—a regular expression pattern that blocks functionality at the Host level cannot be unblocked from the Site level.

Table 12.11 Advanced URL Management Options

Tab	Field	Values	Description
General	Use	Checked/	The lowercase conversion option

	Lowercase URLs	unchecked	<p>converts all URLs generated by the site to be in all lowercase.</p> <p>Administrators often use this site both for aesthetic reasons and to ensure that no search engines regard lowercase and mixed-case versions of the same URL to be considered duplicates of each other.</p>
	Redirect Mixed Case URLs	Checked/ unchecked	<p>When the Use Lowercase URLs option is checked, the Redirect Mixed Case URLs option can also be checked. This enforces the use of lowercase URLs by redirecting any URL that is not lowercase to the lowercase version.</p>
	URL Space Encoding Value	Hex (%20) or +	<p>This option applies to spaces in URLs for the segments of URLs that do not apply to DNN pages. In practice, this applies to third-party generated URLs where the developer does not remove spaces from the URL segments before passing them into the DNN API. By default, the spaces will be removed and replaced with the space encoding value of %20, which is the hexadecimal code used. This option allows the replacement of spaces with the more aesthetic + character, which also represents a space in URLs. However, IIS by default will return a 404 unless configured to allow a + in a URL path.</p>
	Convert Accented Characters	Checked/ unchecked	<p>Accented characters are those characters in the Latin character set that use an accent such as å, î, or ë. These are not represented in the ASCII character set that is legal for</p>

			<p>URLs and will result in encoded page URLs within DNN if used in page names. This option, when enabled, will find any accented character (also called <i>diacritic</i> characters) and replace them with the ASCII equivalent. Thus, å becomes a, î becomes I, and ë becomes e for the generated page URL. Note this has no effect on other Unicode character sets such as Asian or Cyrillic.</p>
	<p>Replace These Characters</p>	<p>List of characters - default &\$+, /?~# <> () ÿ ¡ «» ! " "</p>	<p>The characters in the list represent those characters that will be removed from the list and replaced with the page concatenation character (either - or _, depending on configuration). This list can be expanded with other characters if required. It is important to note that this list starts with an empty space character, which is also replaced. This list applies only to generation of page URLs from DNN page names.</p>
	<p>Find/Replace these Characters</p>	<p>List of character pairs</p>	<p>The Find/Replace list is empty by default but can be used to make URL character replacements. This is done by forming a list of character pairs, where the pair is separated by a , (comma) and pairs are separated by a ; character (semicolon). An example use is the replacement of æ with “ae” for URLs or replacing ß with “ss”. This would be achieved with a string of æ, ae; ß, ss.</p>
	<p>Warn in Log About Duplicate URLs</p>	<p>Checked/unchecked</p>	<p>The flexibility of creating custom URLs within a DNN site creates the possibility of creating two URLs that point to the same DNN page. This</p>

			option activates a check that will detect this scenario and report it within the DNN event Log as a warning.
	Enable Custom URL Providers	Checked/unchecked	A site-level switch to enable/disable extension URL providers from being used. Can be safely unchecked if there are no extension URL providers installed.
Host page only	Allow Debug Code	Checked/unchecked	An installation-level switch that allows a debug code of ? <code>_aumdebug=true</code> to be added to a request querystring, or <code>_aumdebug:true</code> to the request headers. If this value is then found, the response headers will contain debug information similar to that found in the output of the test URLs URL rewriting test.
	Show Page Index Rebuild Messages	Checked/unchecked	If checked, every time the URL index is rebuilt and stored in the cache, a message is logged to the DNN event log. This is used for fine-tuning cache settings and troubleshooting if the page index rebuild process is being run excessively.
Regular Expressions	Ignore URL	Regular expression pattern	This pattern controls which URLs are to be ignored by the URL rewriting process. Any URL matching the pattern will not be processed by the URL Rewriter component.
	Do Not Rewrite URL	Regular expression pattern	The Do Not Rewrite pattern allows URLs to be processed by the URL Rewriter component, but does not allow the URL to be rewritten. This allows important tasks like

			identifying and storing the current site and alias used, without trying to transform the URL to a /Default.aspx?TabId=xx rewritten URL.
	Site URLs only	Regular expression pattern	Each URL processed through the URL rewriting process is first matched against the pages in the site. If no match is found, it is then matched against the list of patterns found in the siteurls.config file. If a URL matches the Site URLs only pattern, it skips the process of matching against known DNN pages and is evaluated against the SiteUrls patterns first.
	Do Not Redirect	Regular expression pattern	If a URL matches this pattern, it will not be redirected by the URL rewriting component. This does not stop redirects generated outside the URL rewriting component or in components external to DNN.
	Do Not Redirect https	Regular Expression pattern	This pattern stops a matching URL from being redirected from HTTP to HTTPS or from HTTPS to HTTP. This is used to stop unwanted HTTPS-related redirect behavior.
	Prevent Lowercase URL	Regular Expression pattern	The Convert to Lower Case configuration option converts all generated URLs to lowercase, and the Redirect Mixed Case option redirects any URLs that are not lowercase. This pattern excludes any URLs from the conversion or enforcement of lowercase URLs. This is important whenever a URL may include case-sensitive values, such as base-64 encoded strings.

	Do Not Use Friendly URLs	Regular Expression pattern	Any generated URL that matches this pattern will use the earlier friendly URL pattern that includes the TabId and page name. It ensures that the canonical URL for a matching page is not the format generated by the advanced URL component.
	Keep In Query String	Regular Expression pattern	When a generated or requested URL matches this pattern, the segment of the URL that matches will be removed from the URL path segments and converted into querystring values. Thus, a match on <code>/key/value</code> will be converted to <code>?key=value</code> on the end of the URL. This is used wherever a segment of the URL must stay in the querystring for proper operation.
	URLs with No Extension	Regular Expression pattern	To provide comprehensive 404 handling, the URL rewriting process identifies any URL it cannot match with a DNN page as a 404. There are many other URL formats that appear to be DNN page URLs but are not. In these cases, a match with this pattern allows the URL to complete processing without being returned as a 404. In practice, these are most often web service calls, as shown in the default pattern.
	Valid Friendly URL	Regular Expression pattern	The inclusion of characters to form DNN page URLs from DNN page names works by not matching this pattern. Any character that does match the pattern will not be included in a generated DNN page URL. This pattern can be changed to include various non-ASCII character

			sets such as the many different Unicode character sets from non-Latin-based languages.
--	--	--	--

[Table 12.11](#) shows the options available in the Advanced URL Management page.

URL Creation for Developers

Custom URLs for DNN pages are created through the Page Settings interface, as shown in the “Creating Custom URLs for DNN Pages” section. This section expands on the concepts of modifying DNN URLs and shows how DNN developers can use the built-in DNN API to manipulate and modify the way in which DNN URLs are used and displayed.

The URL rewriting process converts the page name into a rewritten URL in the format of [example.com/default.aspx?TabID=89](#) where 89 is the TabId corresponding to the DNN page. Updating this field creates a new database record in the TabUrls table. If there is no TabUrls record for a DNN page, then the page uses a URL generated from the Pagename field.

As shown in [Table 12.12](#), page URLs can load the page and all the default view controls for the modules on that page or can load specific modules and module controls. Further, third-party modules can use the URL contents to specify content to load onto the page.

Table 12.12 Creating DNN Page URLs

Type	Example	API Call
Page URL	example.com/page-name	<code>NavigateURL(87);</code>
Page URL with specific module control	example.com/page-name/ctl/edit/mid/876	<code>EditURL(87,"edit",876);</code>
Page URL with third-party content	example.com/page-name/cid/345	<code>NavigateURL(87, "", "cid", "345");</code>
User profile URL	example.com/bill-smith	<code>NavigateURL(65, "", "userId", "34");</code>

The correct way to generate a URL within code is via two different API calls within DNN. [Table 12.12](#) shows the correct calls to use and the results that will be displayed. All the API calls are in the `DotNetNuke.Common.Globals` class.

In [Table 12.12](#), the example values shown in the API Call column point to a hypothetical DNN install and are as follows:

- TabId for Page Name: 87
- Module ID for specific module on page: 876
- Content ID for specific third-party module: 345
- User ID for Bill Smith: 34

When creating URLs for third-party content, the `NavigateURL` call takes a parameter array—these values can be continually expanded to handle multiple combinations, which are then repeated through the generated URL. For example, a call of `NavigateURL(87, "", "key1", "val1", "key2", "val2", "key3", "val3")` will return a URL of example.com/page-name/key1/val1/key2/val2/key3/val3.

Some developers who are new to the DNN patterns make the mistake of constructing the URL for a specific page or resource outside of the `NavigateURL` call. This should be avoided for the following reasons:

- The `NavigateURL` call will take into account any language-specific URL rules that pertain to the URL.
- The `NavigateURL` call will assign the correct domain name (via the site alias) for the URL.
- The `NavigateURL` call encapsulates the URL logic and may change with future DNN versions.
- The URL provider functionality is called within the `NavigateURL` logic and requires all the parameters for a URL to be present within the method call.

The `NavigateURL` call provides scope for developers to control the appearance and subsequent rewriting behavior, but developers seeking to fully customize the appearance and behavior of URLs will eventually reach a limit on what can be achieved. At this point, developers should start work on functionality-specific extension URL providers, which is the subject of the next section.

Custom Extension URL Providers


Extension URL providers are DNN extensions that are used as a plug-in to provide specific URL appearance and behavior. They are ASP.NET providers that are specifically configured to work with both the URL generation and

URL rewriting aspects of DNN URLs, and they connect to the process through a DNN-specific API.

Extension URL providers can be installed as stand-alone components or can be included with other DNN extensions (such as a DNN module) to provide functionality where activated. The applications are numerous—from redefining the look of module-specific URLs to custom redirection and compatibility tasks to modifying the behavior of URLs relating to content that a developer or administrator has no control over.

A full discussion of URL extensions could fill a chapter by itself. This section introduces the use, development, and testing of URL extensions. Interested developers should view the CodePlex project at <http://dnnurlproviders.codeplex.com/>. This project contains working examples, source code, and documentation on developing your own provider.

Installation and Configuration of Extension URL Providers

To install an extension URL provider, use the Extensions page and install the provider using an installation package provided by the developer or vendor. The installation process is the same as any other DNN extension. Once the installation process is complete, navigate to the Admin  Site Settings page and click the Advanced URL Settings tab to open the Extension URL Providers section.

Any installed providers are listed in the Extension URL Providers section. Installed providers are enabled by default, which makes them active immediately. This also requires that the installation-level switch for extension URL providers is active. The impact on the site URLs depends on the individual provider functionality.

To configure an individual provider, click the Edit icon in the list next to the provider. This loads a provider-specific page that provides the ability to modify any specific settings that relate to that provider. Clicking the Update button updates the settings for the provider and closes the settings window.

Testing of Extension URL Provider Functionality

Basic testing of the provider will be to proceed to the DNN page (or pages) where the functionality of the provider is applied. The provider configuration optionally associates the provider with a specific module extension. If this is the case, the provider functionality is applied to all pages that have an

instance of that module on them. A blog module provider would then update the URLs on a blog page to have a new appearance. A representative sample of all the URLs related to that module should then be done to be sure that the provider functionality is working correctly.

Testing Extension URL Providers in Evoq

The Evoq Test URL functionality, located in the Admin ➤ Advanced URL Management page, can be used to test provider functionality. To test the functionality, some knowledge of the “raw” URL used by an associated module is required. This is then entered as the Query String value for the page using the module related to the provider and the URL generated as previously shown.

To test URL rewriting in Evoq, follow the same process as previously explained in this chapter, including the URL as generated by the provider. [Figure 12.6](#) shows test URL rewriting for the Social URL provider that is available on the previously linked CodePlex project page. This provider converts URLs with `GroupId/xx` to use a URL-sanitized version of the social group name.

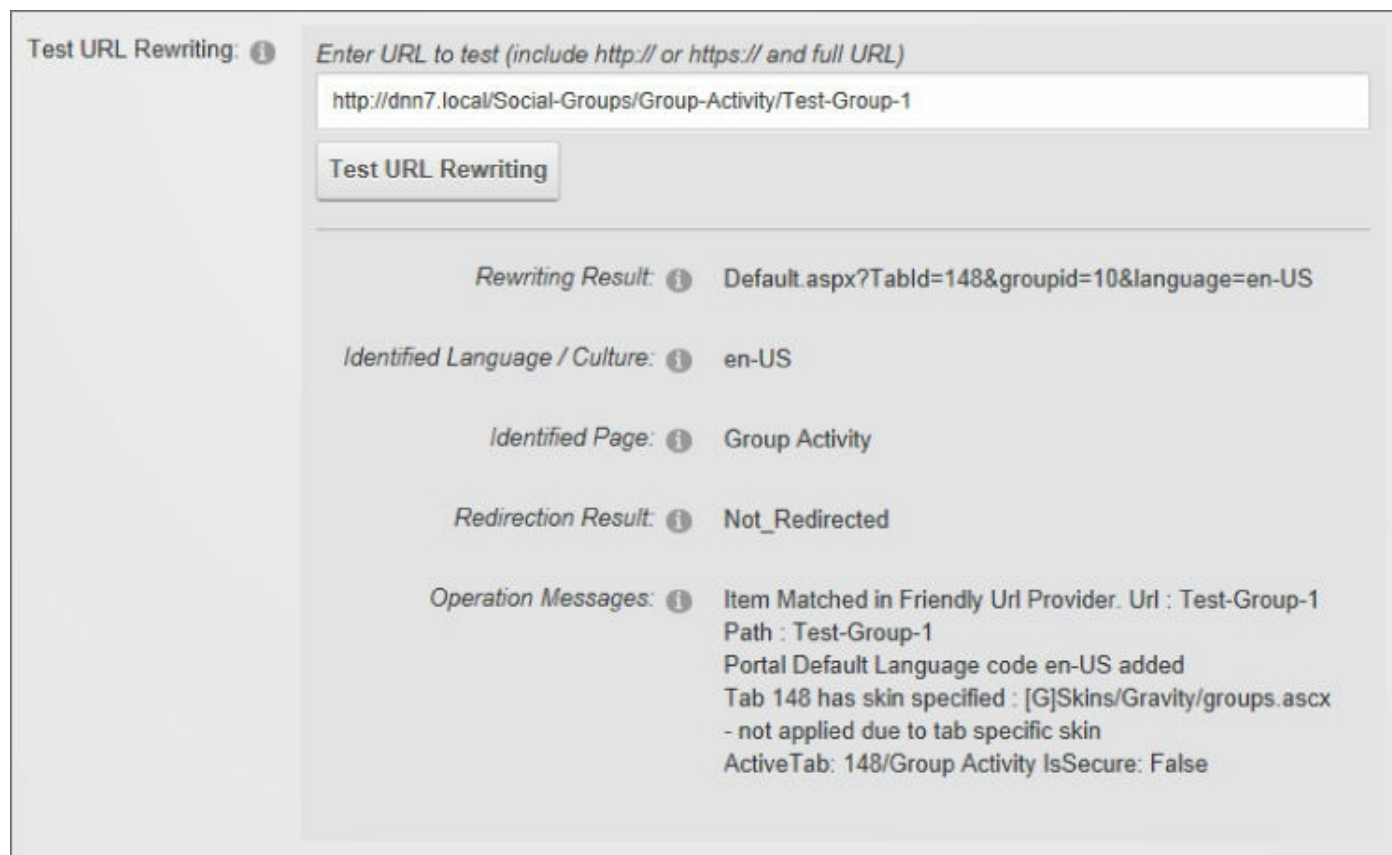
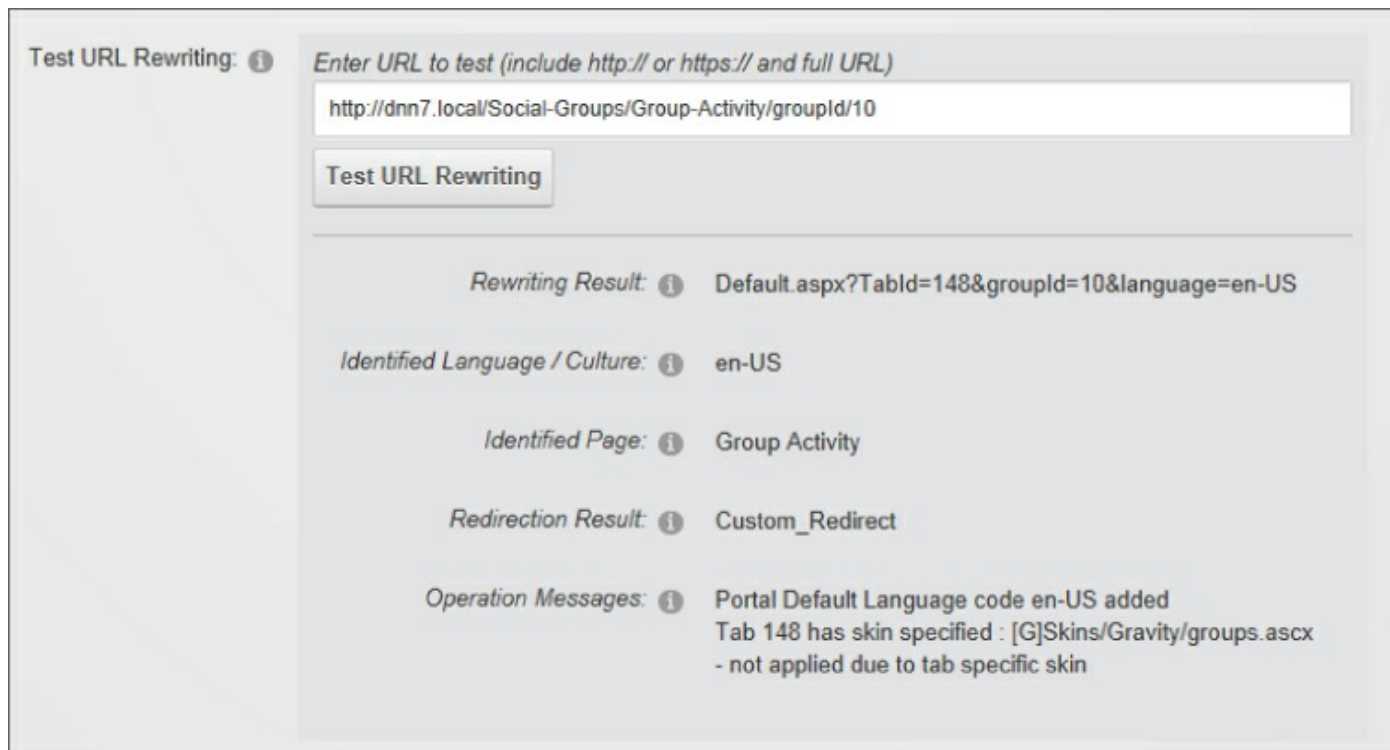


Figure 12.6

[Figure 12.6](#) shows how the requested URL includes the name of the group (Test-Group-1) in the URL, but the rewritten URL includes the Query String value of `&groupId=10`, which is what the underlying Social Groups module is expecting.

Additionally, the Operation Messages section shows that a message unique to the Social URL provider has been added to the output. This shows that the group identified in the URL (`test-group-1`) was matched. This particular message is generated from within the provider itself. Developers can add these messages to help debug behavior during the development process and also for when providers are installed on live sites.

[Figure 12.7](#) shows a similar request, but this time for the “original” URL generated by the Social Groups module before the URL extension provider was installed. This time, however, the result shows that a `Custom_Redirect` value is returned, which means that the resulting URL will be redirected to a new URL containing `/test-group-1` as the URL segment.



[Figure 12.7](#)

This shows how URL providers not only can change the appearance of URLs, they can also be used to redirect legacy and other URLs.

DNN URL Best Practices

Many DNN site owners care about how well their pages rank within search engines for specific terms. Search engine optimization includes ensuring that the site URLs are structured well and work as intended. This section details best practices as followed by DNN professionals to ensure site visitors reach where they mean to go and that search engines can navigate, index, and understand the content on the site.

Configuring URL Options for DNN Sites

New DNN installations come with a range of options pre-configured for URL settings. In most cases, these should be left as is by default. For DNN installations upgraded from earlier versions to DNN 7.1 or later, then all that is required is to activate the Advanced option on the URLs. The default configuration is the same for new installs once the advanced mode is activated.

The following list has some do's and don'ts related to DNN URLs:

- **Do's:**

Use the automatic redirect feature to enforce canonical URLs. This ensures you minimize any content in your site accessible by two different URLs. This problem is called duplicate URLs and should be avoided if ranking well with search engines is important for the site.

Try to preserve the URL of a page that is highly ranked within search engines. A well-ranked page should have the URL kept the same rather than redirecting to a newer URL.

Research and use keywords and phrases for URLs for DNN pages. Customize the URL of the page to match the key word or phrase that you would most like the page to rank for.

Test all URL changes after you have made them. It is very important that visitors don't find broken links or URLs that do not work.

Ensure that every site has a primary site alias selected and that either Redirect or Canonical mode is used to ensure the primary alias is used by search engines.

Carefully evaluate content modules before installing them to ensure that the content created uses a consistent URL scheme that leverages key words in the URLs and doesn't create duplicate content.

Sign up for webmaster services such as Google Webmaster Console and Bing Webmaster Tools to gain an insight as to how search engines index and view your site.

- Don'ts:

Make unnecessary changes to URLs. The URL of a page is key to the visitor being able to interact with the site. All changes should be carefully planned and tested. This is especially important if search engine traffic makes an important contribution to site visitors.

Delete content without providing a redirect to where the visitors should go. This is doubly important when replacing an old site with a new site.

Ignore URL problems—whether it is 404 errors, redirect loops, or a pages not loading problem. It is critical that URL problems are solved quickly before traffic is affected.

Experiment with regular expression settings without a deep understanding of the impacts. Random regular expression changes are almost guaranteed to not achieve the intended goal, but instead impact a site in unexpected ways. Have a plan, test cases, and timing ready, and proceed carefully.

Mix and match redirect technologies. Using a suite of applications to implement site redirects makes troubleshooting difficult. Use DNN for redirects wherever possible.

Ignore the importance of URLs. Many people may question the importance or utility of clean, well-crafted URLs and the absence of unwanted redirects and other problems. The URL is the gateway to visitors, and as a book is judged by its cover, the webpage is judged by its URL. This is important to both search engine bots and indexes, as well as CPM and CPC marketing, posts in social media, and even the humble URL in an email.

Troubleshooting and Debugging DNN URLs

Using the Evoq Test URLs section was covered earlier in the chapter. This is a very useful tool for developers and administrators working with URLs but cannot replace real-world testing with the actual site in action. URL problems are notoriously difficult to troubleshoot because a nonfunctioning URL often

doesn't show error messages or even logs. To properly troubleshoot a URL problem, you need to see the HTTP traffic for each connection and response.

The best tools for this purpose are those that show each individual HTTP request/response and what the status codes, request/response headers, and returned stream are. Here are some helpful tools for URL troubleshooting:

- **Fiddler:** An HTTP debugging tools that captures HTTP requests and responses. See <http://www.telerik.com/download/fiddler>.
- **Wireshark:** An HTTP debugging tool similar to Fiddler for debugging HTTP requests and responses. See <http://www.wireshark.org/download.html>.
- **Chrome Browser Tools:** Built into the Google Chrome browser. Press F12 to open the Browser Tools and click the Network tab to see HTTP requests/responses.
- **Internet Explorer Developer Tools:** Browser-based HTTP debugging tools, also under the Network tab.

The essential task in debugging URLs is to follow the relevant HTTP request for the DNN page (or other site resource, such as a web service call or JavaScript file). This will respond with a specific HTTP Status code, which in most cases will be 200 for OK, indicating the resource was found on the web server and sent back as normal. The advantage of using a tool like Fiddler over the browser-based tools is that Fiddler maintains a back history of all requests going through the Internet connection. Browser tools generally display the requests only for the current page. This makes it harder to diagnose problems with redirects, where the redirect means a new page is loaded in the browser. When this happens, the history of the previous requests is lost.

[Table 12.13](#) shows common HTTP response codes and their meanings.

Table 12.13 Common HTTP Response Codes

Response Code	Name	Meaning
200	OK	Resource was found and is being returned. The nature of the response depends on the request.
301	Moved Permanently	The request found a resource that has been permanently moved to a new URL. The new URL is

		provided in a response header called Location. Search engines use the 301 status to update their indexes when a page or resource returns a 301 request.
302	Found	The URL is found at another location. Commonly thought of as a “temporary redirect.” Search engines typically do not update their indexes when a 302 redirect is encountered, as the implication is that the originally requested URL may start working normally again. Also returns the new URL for the content in the Location response header.
304	Not Modified	A status returned by a web server to indicate that the resource requested hasn't been modified since the last request. Used by web servers to locally cache static resources, typically for images, script files, and other similar objects.
400	Bad Request	This response code means that the web server could not understand the request. It may mean a malformed request (request headers incorrect) and can sometimes mean a malformed or illegal URL.
404	Not Found	Returned when the web server cannot locate any resource that matches the requested URL. Often paired with a specific “The content could not be found” HTML page that a browser will render.
500	Server Error	Returned when the request was accepted but the server encountered an error while generating a response. This may be related to the URL (the URL may contain segments that are legal but the software cannot process) or may be related to something else entirely, such as a failed database connection.

To troubleshoot URLs, it is important to understand what HTTP status code is being returned by the URL.

Follow these steps to debug a DNN URL using Fiddler:

1. First, re-create the test case for the URL using a normal browser on your

computer. This may be as simple as opening the browser and entering the URL, or it may require logging onto the DNN site and performing a specific action like opening a pop-up window.

2. Open Fiddler on your computer and check to see that the lower-left corner of the Fiddler window shows the message “Capturing.” If it does not, double-click the status rectangle, or press F12 to start capturing.
3. Redo the action that causes the problem you are investigating. As you use the browser, Fiddler is filling up the left-hand list with each HTTP request/response pair.
4. Once you re-create the problem, stop capturing traffic again by pressing F12 in Fiddler, or double-click the Capturing message. This prevents the list filling up with noise as other services on your computer send HTTP traffic.
5. Search the HTTP sessions in the list, and select the one that matches the URL you are trying to debug. This may be obvious if you are just requesting a page. It may require careful sorting if you are looking for the URL that belongs to a certain image, pop-up, or web service executed while using the page.
6. Once the problematic HTTP session is highlighted, click the headers on both the upper and lower-right panes in Fiddler. This shows the request headers on the top and the response headers on the bottom. The top of the response headers will show the HTTP status (such as HTTP/1.1 200 OK or HTTP/1.1 400 Not Found).
7. Look into the response headers section to find if there are any specific messages from DNN. DNN may add response headers providing further information, such as why a 404 or 301 status code was returned.

Using the Debug Mode for Advanced URLs

DNN includes a debug mode in the Host Settings page. Once this mode is activated by checking the check box and updating the Host Settings, the DNN URL rewriter actively looks for either a querystring parameter or a request header. The querystring parameter is used by adding `?_aumdebug=true` to the end of a requested URL, and the request header by adding `_aumdebug:true` to the HTTP request. In both cases, this triggers an internal trace on the URL rewriting request, and extra response headers will be added to the output. The

number of responses will vary, depending on the request, but generally the same information is available through the test URL functionality in Evoq. [Table 12.14](#) shows what this information means—it is output in a single line with commas separating each value. An example debug response header appears as follows:

```
http://dnn7.local/?_aumdebug=true, , Default.aspx?TabId=56&_aumdebug=true&language=en-US, CheckFor301, 7.2.0.563, 0:dnn7.local, Normal
```

Table 12.14 URL Debug Response Header Data

Name	Example Value	Meaning
Requested URL	http://dnn7.local/?_aumdebug=true	The URL as originally requested by the visitor's browser or device.
Redirect Location	http://dnn7.local/new-location (not shown in above example)	If the URL results in a 301 HTTP status, the location of the new URL where the request will be redirected.
Rewritten URL	Default.aspx?TabId=56	The rewritten URL as specified by the URL Rewriter. This is the format of the URL as the rest of DNN will see it.
Action	CheckFor301	An internal action code that determines how the request should be handled. CheckFor301 means to check to see if the request should be redirected. In this example, it was not redirected.
Version	7.2.0.563	The current version/build of DNN against which the request was issued.
Portal ID / Site Alias	0:dnn7.local	The portal ID of the site that was identified by the request and the site alias that was identified for the request.
Browser Type	Normal	Either Normal or Mobile, depending on the response from the configured browser detection component.

Further response headers may appear, which may include messages from the URL rewriting process (such as a redirect reason) or responses from configured and active extension URL providers. Again, these are the same messages as seen in the Evoq Test URL section. The difference is that these

are shown from the actual request running, rather than through the test mode, which doesn't actually make a real HTTP request to the server.

A production server should not be left in Debug mode, because it will have a slight impact on performance and because it reveals internal data about the installation to give malicious actors vital internal information. Debug mode should be used only for production sites for enough time to collect the data and diagnose a problem and then switched off.

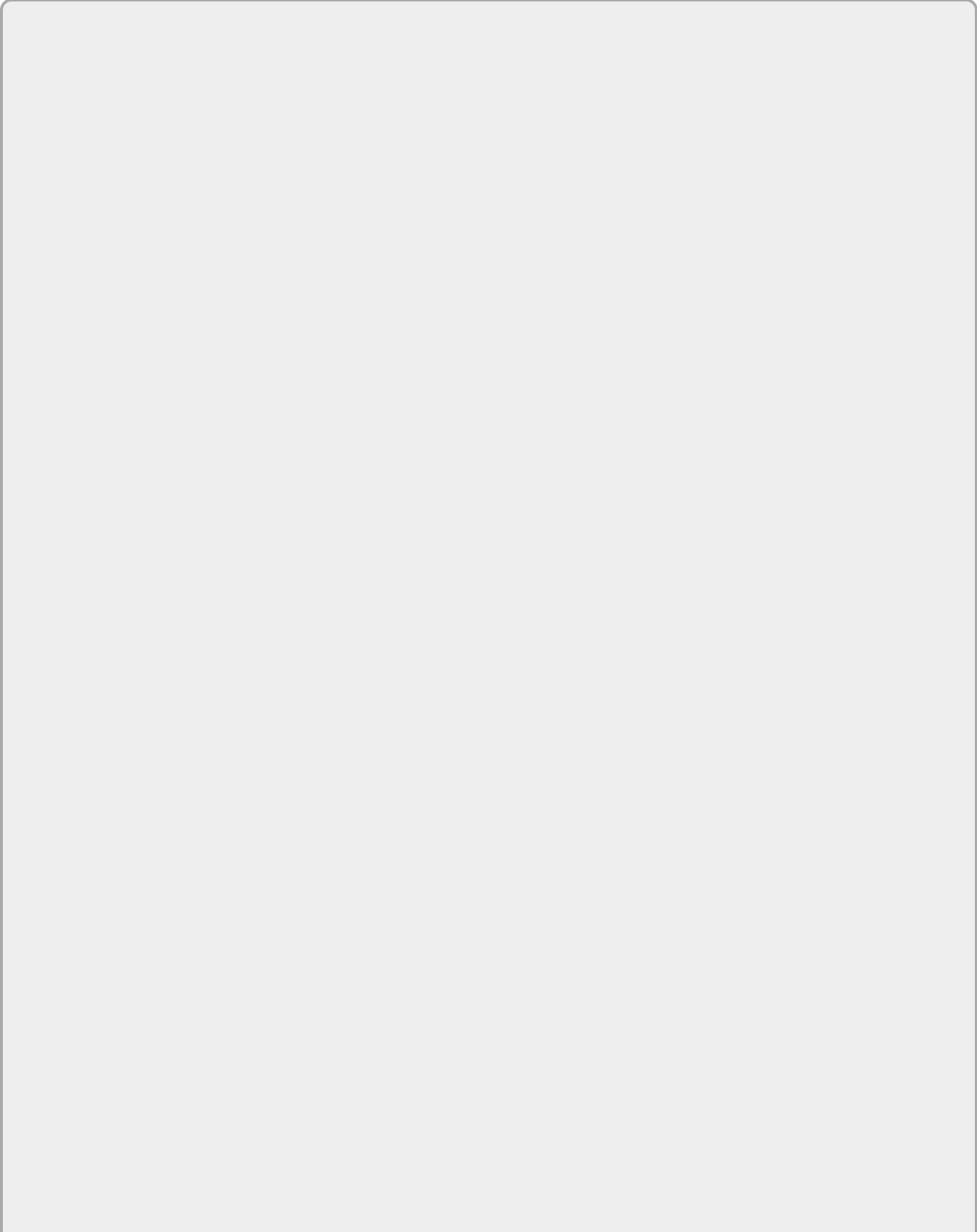
Summary

In this chapter, you learned the history of DNN URLs as a way of understanding how and why the friendly URL functionality has been designed for DNN 7. The chapter covered the activation of the Advanced mode of friendly URLs and how they apply to SEO for DNN sites. The chapter covered the different URL configuration options and how they affect the behavior of DNN.

You learned how to create custom URLs for DNN, how to generate URLs within DNN modules, and how to configure and use extension URL providers. Finally, the chapter covered best practices for URLs within DNN and how to troubleshoot and debug problems with URLs in your DNN installation.

Chapter 13

Beginning Module Development



WHAT'S IN THIS CHAPTER?

- Choosing your tool set
- Setting up your development environment
- Understanding design and embedding of modules
- Creating a module online in DNN
- Creating a simple Razor-based module
- Creating a more complex module
- Distributing a module

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code comes as two downloads of the finished sample module you'll be building in this chapter:

- Guestbook_01.00.00_Install.zip
- Guestbook_01.00.00_Source.zip

As previously mentioned, the number 1 feature of the DNN Platform is its extensibility, notably the ability to easily install and manage modules. A module comes as a Zip file. You upload this file into the platform, and DNN takes care of the rest whether it's the first time you've uploaded it or whether it's an upgrade. The part of the platform that takes care of this is aptly named the *Installer* and is discussed in more detail later. The takeaway here is that you can leverage this technology to create your own modules that are highly transportable. And just like Apple's App Store, DNN also has its DNN Store (<http://store.dnnsoftware.com>, see [Figure 13.1](#)), and maybe one day you will sell your module there. But first you need to master module development. And although it's not trivial, there are different ways to do this requiring different skill levels. To get something working in a relatively short timeframe, in this chapter we give you the background you need to know *where* to start and *how* to create your first module.

Register | Login | dnnsoftware.com | Newsletter Sign-up

DNN STORE

Search for vendor, product or keyword MY CART: (0) items total: \$0.00

Modules Themes Featured Make A Website Make Money About Us Help Center Blog

Modules

- Communication
- Content Management
- Functionality
- Images & Media
- Marketing & Social
- eCommerce
- Mobile
- All Types

Themes

- Colors
- Genres
- Types
- Singles
- Packs

Featured

- Top Modules
- Top Themes
- Top Vendors

Make A Website

- Type of Site
- Solutions

THE FLEXIBILITY OF DNN + ENTERPRISE E-COMMERCE

Hotcakes Commerce is built on the MVC Framework. No new language to learn. No limits to your design.

[LEARN MORE](#)

Top Selling Modules

<p>Document Exchange 6.1 <i>Bring2mind</i></p> <p>\$249.00 > Learn More</p>	<p>DigBundle 1 Year Subsc.. <i>digNuke</i></p> <p>\$149.95 > Learn More</p>	<p>AJAX FAQ 2.6 (Multi L.. <i>DNNCentric</i></p> <p>\$39.95 > Learn More</p>	<p>The Reservations Modul.. <i>DNN Specialists</i></p> <p>\$99.00 > Learn More</p>
--	--	---	---

Top Selling Themes

<p>Porto <i>Mandeeps.com</i></p> <p>\$149.95 > Learn More</p>	<p>EDS Theme and Module Collec.. <i>EasyDNNSolutions.com</i></p> <p>\$149.95 > Learn More</p>	<p>Unlimited DNN Theme 033 (V3.. <i>dningo.net</i></p> <p>\$149.99 > Learn More</p>	<p>Unlimited DNN Theme 029 (V3.. <i>dningo.net</i></p> <p>\$149.99 > Learn More</p>
--	--	--	--

Figure 13.1

There are many ways to do this. The permutation of tools and technologies available makes it impossible to tell you how you should develop a module. It is a matter of personal taste. This chapter gives you the background to make those decisions. You'll see what a module is in technical terms, how a module is embedded in DNN, and what is involved in packaging. We use several

examples to show you how this can be done to give you a start in this exciting technology.

A Guided Tour of Your Work Environment

First, allow us to guide you as you look at ASP.NET applications, the DNN Platform, and DNN modules. It's important to understand how these various bits are built and relate to each other.

ASP.NET Web Forms

If you already know how to program an ASP.NET Web Forms application, you can skip this section. ASP.NET Web Forms was introduced in 2001 with the first release of ASP.NET. Microsoft aimed to provide programmers with a programming model that resembled Windows programming on the web. For that it needed to resolve the web's stateless nature. This means the server is not connected to the client outside of simply delivering some files, and every time a browser makes a request it is seen in total isolation. The server does not “know” you made a request just 2 seconds before. This was not an issue in the old days of the static web. After all, all the server did was serve files on its hard drive to the client browser. But now Microsoft wanted to make it appear to programmers as if there was some kind of continuous connection between the browser and the server. It solved this by passing a whole lot of hidden data back and forth between the browser and the server. Basically, every time the browser made a request it would send enough information to the server so it would know what happened previously. Although this mechanism has come under increased attack due to the volume of data being sent back and forth (for example, slower response times), it remains a powerful paradigm for web programming and easy for developers to understand.

So Web Forms behaves a lot like Win Forms in that you have a surface onto which you drop components (text boxes, buttons, drop-downs, and so on), and in the so-called *code behind* files of these forms you catch events that are triggered when a user interacts with those components (for example, clicking a button, filling of a text box, and so on). The aforementioned form is essentially a template where HTML is intermingled with tags that the Web Forms processor recognizes and turns into what you need it to be. These templates are easily found on your server's hard drive as they have either the .aspx or the .ascx extension. An .aspx is a standalone web form, intended to be served as a complete page to the browser. An .ascx is a partial file called a *user control*; that is, it is embedded in either an .aspx or another .ascx (you can nest them) and is similar to the other components like text boxes and buttons

that you add. The code behind files are either .vb (Visual Basic) or .cs (C#).

If you now go to your installed DNN application, you'll understand what the majority of files are about. Other files you find are mostly static files (that is, they are meant to be served "as is" by the server to the browser) such as images, CSS, and JavaScript files. But there remain some others that are not served and are meaningful only to the ASP.NET engine. The first we want to call out is the *web.config*. It contains application-level configuration and sits in the root directory. It is an XML file that tells IIS various details about the application. Most importantly it tells IIS to which components to hand requests (routing). It is also the place where some application-level settings are stored, such as the connection string to the database. This is extensible, and various bits of the application can also use the *web.config* to store settings (DNN stores information about its configuration in this file as well). You should not use this file trivially. Although it's perfectly legal for you to use the *web.config* to store settings for your module, keep in mind that the *web.config* in many installations may have been changed by the owner, and your settings may be inadvertently tampered with. As a general rule, try to avoid tampering with this file as you can easily break the entire application if something goes wrong.

Finally, pay attention to the bin folder. That is where the .dll files reside. DLLs are compiled libraries of code. There are two ways to run code on the server in ASP.NET. You can have code files on the server that are compiled on the fly—that is, when a request comes in. Or you can precompile code into libraries and drop those in the bin folder.

There are several advantages with the latter approach. For a start, it means that the server doesn't need to compile because it has already been compiled. This is offset by the fact that the server only compiles if it finds that any of the code files have been altered. But the compiled library offers a performance advantage.

Second, the compiled library hides the code from prying eyes. This may be important if you deliver a commercial solution and you don't want to hand over your source code. It also shields the code from tampering. This might be useful in cases where others have access to the server and you want to make sure no one tampers with the code.

Finally, it makes it simpler to manage versions of your work. A DLL can be labeled with a version number. Code files can be labeled this way, too. But

needing to check if any code files may have been updated can be a cumbersome task, and having all code in a single DLL is just that much simpler.

Many of the DLLs in the bin folder are interdependent. So you can't remove or replace them without risk of breaking the application. In .NET the dependencies between DLLs are recorded when they are created. For example, DotNetNuke.Web.dll version 7.3.4 is dependent among (many) others on DotNetNuke.HttpModules.dll version 7.3.4, DotNetNuke.Instrumentation.dll version 7.3.4, DotNetNuke.dll version 7.3.4, DotNetNuke.Web.Client.dll version 7.3.4, and DotNetNuke.WebUtility.dll version 4.2.1. If any of those DLLs is replaced by one with a lower version or removed altogether, DotNetNuke.Web.dll throws an exception and halts all its work, breaking the entire application. So tampering with DLLs is not for the faint of heart. You really need to know what you're doing.

This concludes the brief overview of some of the critical parts of an ASP.NET Web Forms application. Be aware of this in case you want to change things or if you need to debug some erroneous behavior.

The DNN Platform

As a Web Forms application, DNN also has a bin folder, a web.config file, and many of those .aspx and .ascx files. DNN (or DotNetNuke as it was called at the time) came right on the heels of the emergence of ASP.NET and was a continuation of one of the bigger sample projects in the first wave of ASP.NET. In many ways, DNN has been a trailblazer for Web Forms, and at times Microsoft has adjusted what it was doing based on what was happening in the DNN ecosystem. Those who have been around since that time can see that history reflected in the various files and folders of the application. For newcomers, it can be puzzling why you'd have /admin and /DesktopModules/Admin. Or /js versus /Resources/Shared/scripts. Over time, many developers have worked on the solution, and like binary stars, DNN and ASP.NET Web Forms have evolved in each other's gravity, sometimes changing the other's course. So at times the anatomy of DNN may feel somewhat quirky. With that out of the way, let's look at what you have on your hard drive after installing DNN. [Table 13.1](#) shows a rundown of the most important folders in the application.

[Table 13.1](#) Major Directories in a New DNN Installation

Path	Description
/admin	Various bits of functionality used throughout the framework, like the login button, the control panel, or the module's settings panel.
/App_Browsers	Configuration files that allow DNN to know what a browser's capabilities are.
/App_Data	This is the standard ASP.NET folder to which an application's database will be written if you don't specify otherwise.
/App_Data/Search	DNN uses Lucene to index its contents. Lucene stores its files here.
/App_GlobalResources	You find DNN's common resource files here. These files store the most common text used in the application (like "Cancel" or "Submit").
/bin	See the "ASP.NET Web Forms" section of this chapter.
/Components	This is mostly empty. It used to be the place where the source version stored its source code.
/Config	The default configuration files for various aspects of the application (the currently applied files are in the root folder). These files remove the need to put everything in the web.config file.
/controls	Reusable controls. You can use these in your own modules as well. Controls like the

	UrlControl (choose a URL or a page in DNN) will save you a lot of work.
/DesktopModules	This is where modules are stored.
/DesktopModules/Admin	Administrative modules like the SQL module or the Extensions module. You'll see these modules in action on the various admin pages.
/Documentation	Licenses and a quick guide to installing DNN.
/Icons	Icon image files that are used in the application.
/images	Various images consisting mostly of the icons that were used in older versions of DNN.
/Install	When DNN first installs, it uses these folders to create the system. The default modules are found here as well as default skins and languages, for example. If you create a distribution based on DNN, you put all your own stuff in this place so it installs by default.
/js	Various JavaScript files.
/Licenses	More licenses of components used in DNN.
/Portals	Data directories that hold the files for the various portals/sites.
/Portals/_default	Data directory for the host. This is where you find the site templates (the files that determine what pages to create

/Providers	for a new site), for example. Directories dedicated to extensible bits of DNN. You can override the default providers with your own if you want.
/Providers/DataProviders/SqlDataProvider	The default data provider is the SQL Data Provider. This means we use SQL server by default. This directory contains all of the scripts DNN uses to get the data schema installed correctly and fill it with the first necessary data.
/Resources	Various shared files that can be served depending on the context. Modules may request jQuery to be sent to the client. The jQuery scripts are found under this folder. The folder has grown significantly since version 5 of DNN, as more logic has been transferred to the client side. And the folder structure reflects the strong growth in that not everything is organized 100 percent logically.

There is a certain analogy between the DNN folder structure and a typical Windows machine's folder structure. The DesktopModules folder is like the Program Files folder on your Windows machine. It stores the various installed programs. The Portals folder is like the Users folder. It stores the user's files and makes sure some level of isolation is used to prevent one looking into the files of another (site folders). Finally, the /bin folder is like the System32 folder in that the DLLs that are shared across the whole application go there.

Anatomy of a DNN Module

Now that you know what an ASP.NET Web Forms application and specifically DNN consist of, let's examine the constituent parts of a DNN module. As previously mentioned, a module is delivered as a Zip file. But what's in the Zip file and how does this fit into DNN? To better understand this, you begin by dividing your application into the age-old split of data and code. The code comes as .NET Web Forms files such as .ascx, .js, .dll, or code files (.cs, .vb). And these files are almost without exception stored in a subfolder of DesktopModules in your DNN installation and the bin folder in case of a DLL. The user's data comes in two forms: files and relational data (that is, tables of data). File-based data is stored in the site home directory (default Portals/[PortalID]/) and SQL data in the SQL Server database. For the former, you won't need to do anything (the site home directory already exists), but for the latter you will need to create the necessary tables, procedures, and so on, yourself through a set of scripts. These scripts create or update the schema your application needs in SQL. That's it. What you create is a bunch of files that either are copied to the DesktopModules folder or the bin folder or are used to generate the schema in SQL that the application will use.

Let's look at the HTML module to see what this looks like in practice. In your installation you should find the following directory: /DesktopModules/HTML ([Figure 13.2](#)). There are four .ascx files and one .css file in that directory.

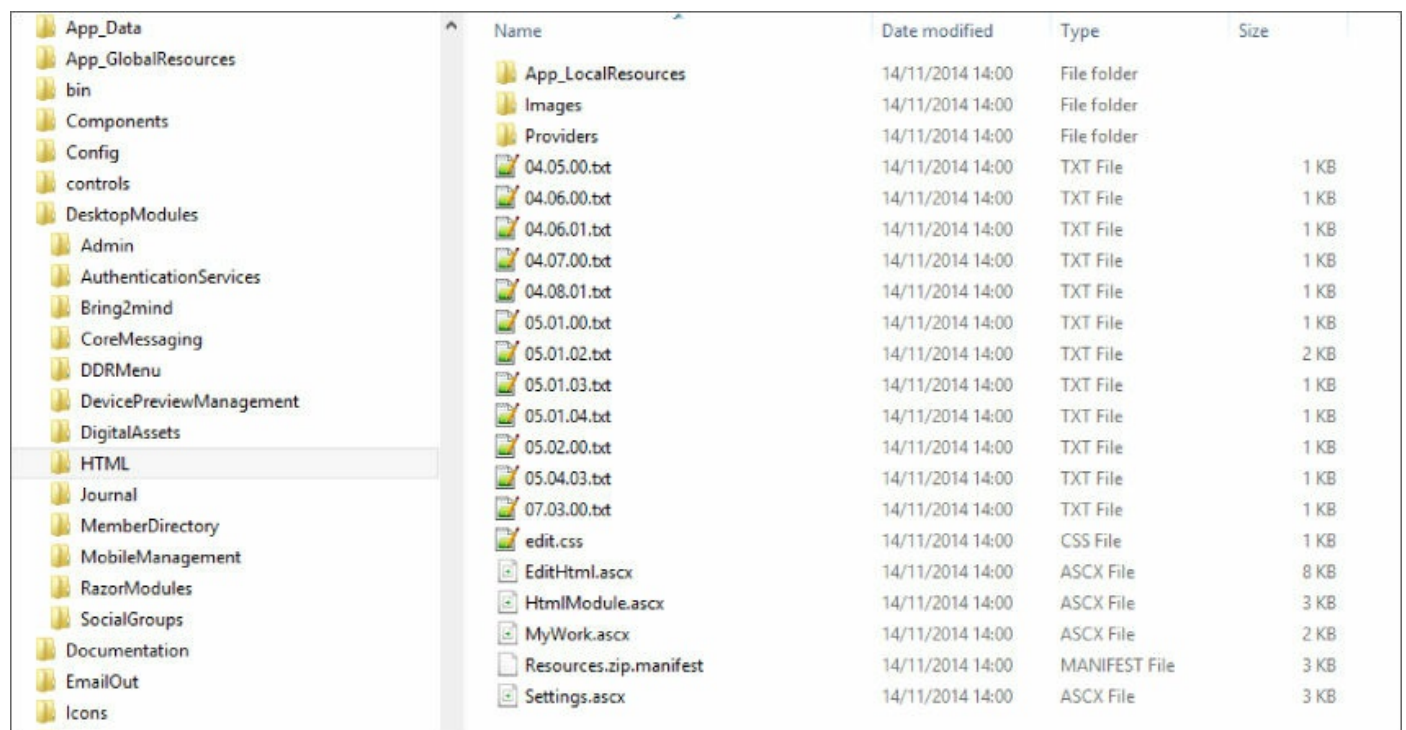


Figure 13.2

These files are actually used to render to the client what the developer wanted (there are also a bunch of .txt files and a .manifest file in this folder that you can ignore for now). Then there is an App_LocalResources folder. Like App_GlobalResources, this holds resource files that contain the text snippets that are used in the application (more information about this in the “Localization” section). Under Providers/DataProviders/SqlDataProvider are all the SQL script files that the module uses. These script files are used only during installation, so if you tamper with them now, nothing changes. They remain on hard drive as a record of what has been done.

The only file you're missing from the module is the one that was written to the bin folder: DotNetNuke.Modules.html.dll. That file is the compiled version of all the source code that came with the module. In the source code version of DNN you'll find this source code under DNN Platform\Modules\HTML. This directory holds the entire HTML module project and is used to generate the Zip file that is used to install it.

Your Toolbox

There is probably nothing more personal in programming than your toolbox. Every time we exchange ideas with another developer, there is some moment in the conversation that will slip into a “you should use X for that” type of discussion. We don't think we've ever encountered someone with exactly the same toolbox. So whatever we mention in this chapter, there are always alternatives and some will never even use a similar tool, and we're sorry if we didn't mention product X. So we will just mention a few tools that we believe just about everyone is using and then you can figure out what else you want to use. Note that there are excellent resources you can use to find out more about tooling. Scott Hanselman is a very well-known .NET technology evangelist and is hugely influential in the .NET ecosystem. Next to blog posts, podcasts, and the like, he maintains the very useful “Ultimate Developer and Power Users Tool List.” If you want to find new ideas for your toolbox, visit the list at <http://www.hanselman.com/tools> and check out the developer section.

Your primary tool for development will be your IDE (Integrated Development Environment). And the default .NET IDE is Microsoft's Visual Studio. Visual Studio comes in various flavors, some commercial and some free. As this book was being written, Microsoft has launched a new free version of Visual Studio called “Community” that replaces the previous “Express” labeled versions. From what we've been able to see, there is now a completely free and unbeatable IDE for all of us, whether beginner or advanced programmer. You can also go open source with SharpDevelop, or you can go totally bare-knuckled with your text editor of choice, but in this book we use Visual Studio.

Then you need some way to access and manipulate the database. You can run scripts in DNN using the Host SQL module, but it doesn't replace the ability to craft queries in an editor and run them instantly. For that you need SQL Management Studio. There are probably alternatives out there, but it comes free with SQL Server (Express), so we never looked further.

Finally, there are many tools available that you can add to make your life easier, but it really depends on your level of module development. To start with those listed previously is fine, but as you advance you may want to look at source control (Git and SourceTree are favorites) and HTTP debugging (Fiddler). The latter allows you to examine the HTML traffic between the

server and the browser. This is extremely useful when debugging Ajax calls, for example. In fact, it's the only way we know to effectively debug this scenario.

The Environment

It's time to set up the environment in which you'll create your module. This means having a DNN instance up and running that you can access directly using Visual Studio.

DNN

For our environment we use the regular DNN Platform distribution that you can find through the download link at <http://www.dnnsoftware.com> or directly on CodePlex or GitHub. You'll typically find several distributions: the *New Install* version, the *Version Upgrade* version, the *Source Code* version, and the *Web Platform Installer* version. The only difference between the first two is that the upgrade version comes without a few files that are typically tuned to your specific setting (like the web.config). So you can safely unzip and drop all files from an upgrade version over an existing version of DNN, and the first time you go to the site, it will upgrade the application internally. The web platform version is meant to work in conjunction with Microsoft's Web Platform Installer.

We use the regular version for our development environment because we don't need to run the source version of DNN in order to develop a module. When developing our module, we treat DNN as a black box and only use its API. Unless you want to find out how DNN works internally, you don't need to develop inside the source version of DNN. Another good reason to use the nonsource version is that it is faster. The source version requires a bit more work from the .NET compiler, and it quickly will become tedious if every time you want to test a code change you need to wait a minute before the site fires up. Finally, another reason for steering clear of the source version is that you'll want your module to work in a production environment somewhere. And that is where the nonsource version will be running. So you'll mimic as closely as possible the end result while developing your application.

Which DNN version should you choose? This depends largely on whether you intend to distribute your work for a wider audience or whether this is a one-off module where you already know the environment where it's supposed to run. Obviously for the latter you would choose the version on which it's intended to run. But if you want to distribute your module for sale or for free, keep in mind that not everyone is running the latest version of DNN. So you might want to backtrack a little. The downside, of course, is that older

versions have fewer features. It requires some level of experience in module development and good familiarity with the DNN API to make this call.

The numbering of DNN versions follows the pattern of other .NET applications in a familiar x.y.z pattern, where x, y, and z are integers. By convention we write this with double digits, so 7.3.4 becomes 07.03.04. The first digit is the *major* version number, the second the *minor* version, and the final digit is called the *revision*. The idea is that the risk of breakage (that is, methods in the API no longer working the same as before or changes in the SQL schema) is higher for changes in the major version number and smallest for the revisions. The latter are sometimes referred to as *bugfix releases* because they don't add or change any features but make repairs to new features that were added in the .0 version. So 7.3.4 is the fourth bugfix release of the 7.3 release. You'll find that commonly the .0 releases are rarely used in production environments because administrators wait to see if any bugs were introduced with the new features. .0 releases are more frequently used by developers to begin using the new features. Keep this in mind when choosing the version on which you'll develop.

[Table 13.2](#) shows a list of highlights of releases since version 6.0. This may help you determine which version is the best for you to target if you aim to redistribute your work.

[Table 13.2](#) Major DNN Releases between 2011 and 2014

DNN Version	Release Date	Highlights
6.0	July 2011	Conversion from VB to C# New UI with pop-ups Azure compatibility Integration with DNN Store
6.1	November 2011	Client resource management (Mobile) Device Capability API Site redirection (target mobile) Site groups (Pro Edition)
6.2	May 2012	Social API (groups, journal, social authentication, and so on) MVC Web Services support (later pulled in favor of Web API)

7.0	November 2012	New (simplified) installer New UI and control bar Content sharing (Pro) Version comparison (Pro) Support for Web API
7.1	July 2013	New Search API New URL management Changed to hashed passwords
7.2	December 2013	Inclusion of Bootstrap into default skin JavaScript library management Subscriptions and digest notifications
7.3	June 2014	Performance enhancements Remote site home directories

SQL Server

Just as with Visual Studio, SQL comes in commercial and free (labeled “Express”) flavors. And just like with Visual Studio, the free version suffices for your task at hand. In a nutshell, the SQL Express versions are fully compatible with the full versions of SQL but just have some size and performance limitations that don't really concern you. As you develop, you want to see your code run and know it runs well in production.

As to the version (2005, 2008, and so on), it is wise to stick to the version that DNN requires. This ensures you won't be doing anything that won't run on someone else's installation. The dependency was moved up to 2008 in version 7 of DNN. If you're just using tables, views, simple procedures, and functions, then it doesn't really matter whether you use 2008 or 2012. But you should be aware that some of your future users might be on 2008.

IIS

You'll need to run your site, so you need Microsoft's web server IIS. There are two flavors of this: a developer version called IIS Express and the full-blown regular IIS. If you have a professional version of Windows, you have access to a full version of IIS. Choosing the right one is not trivial and it has implications for your project, notably regarding debugging. In order to debug, Visual Studio has to somehow be able to piggyback onto the process that is running your application so that it can monitor what is going on. So let's look

at the differences.

The regular IIS is integrated into your Windows environment and runs as a service. The obvious advantage is that it is exactly the same product as you find on a Windows Server edition, so you have peace of mind that what you observe you'd also observe in production. The downside is that Visual Studio doesn't really penetrate this. It can't go through the front door so to speak and ask to piggyback on the so-called *worker process* (the process that runs your DNN and module is called the worker process and has the executable name *w3wp.exe*). Actually, if it could do this, it would constitute a grave breach of security. So to be able to debug your module, Visual Studio has to jump through some hoops. Instead, it needs to ask Windows to be able to latch onto the worker process. But this is possible only if you are running Visual Studio with administrator privileges. Either you turn off UAC on your machine or tweak your Visual Studio shortcut to run as administrator by default.

IIS Express was introduced with Visual Studio 2010 and replaced the older IIS developer edition called Cassini, which really didn't compare with IIS. The new IIS Express was made to behave much more like its big brother. The biggest difference is that IIS Express is a standalone application, so it does not run as a service on your machine. Instead, it's Visual Studio that boots it into action when you press F5 in your web project. Your project's settings will configure IIS Express to serve up the site you're working on, and the process remains active until you stop debugging. Obviously, Visual Studio has no issue latching onto the process here as it starts it up itself. There is also no issue with UAC as you run the process in your own account on the machine. So there are no hoops to jump through. The downside, however, is that all this works only when firing up a website. So to do this for a module project, you need to at least also include the website project DNN (still you don't need the source version!) in your solution and set it as a start-up project. There is another complication in that when you start your project this way, it fires up your default browser using localhost:[some port number]. Under normal circumstances this then adds itself to the aliases of the running site. But it becomes impossible/very complex to create new sites in the running instance as you need to register new URLs, and IIS Express can't do that. Finally, the moment you stop debugging you kill the process. This means you cannot make any code changes while the site is running. You don't have that limitation if you run the regular IIS.

In conclusion, IIS Express is a great tool and a big step forward since the old Cassini days, but it was really designed for monolithic website projects (where you're creating a whole website). For the purpose of developing a module inside a website that you don't want to touch, it is too limiting, and it is preferable to use the full-blown IIS if you have access to it.

To VM or Not to VM

You can go one step further to emulate the situation where the module will end up by creating a virtual machine with a complete Windows Server edition on it. This comes at no cost if you are an MSDN member. The upside is that there are no more differences possible between your development environment and the production environment. Another benefit is that your development environment becomes trivially easy to move to another machine and to back up. Naturally it also comes at a cost. There's some performance loss, as your PC is running two operating systems side by side. Plus you need to install your development tools inside the VM.

Organizing Your Project


The number of options available to develop a module for DNN can be overwhelming. There are so many ways to do this that it's easy to get lost. As ASP.NET and DNN progressed, new methods evolved over the course of the last decade to make things easier for you and to give you more options. And similar to the choice you make for your toolbox, many decisions come down to personal preference.

Inline versus External Module Creation

You'd be surprised what you can do without any tools. DNN includes a number of ways in which you can create a module inline in the application. As modules are “active components,” you need to be logged in as superuser. If you switch to Edit mode and you hover over any module's menu, the Develop menu item appears. This allows you to edit any of the module's controls directly inside your browser. DNN includes a powerful code highlighter called CodeMirror to help you. Combined with the ability to create modules from scratch on the Extensions page (through the Create New Module button), this means you can create an entire module without ever going to the files directly.

Hello World

Here we'll create a module using the inline module creation abilities of DNN:

1. Open your DNN site, log in as host, and go to Host  Extensions.
2. From the buttons at the top, click Create New Module.
3. Select New in the Create Module From drop-down list.
4. Click the Add Folder buttons to create an owner folder called “WROX” and a module folder called “HelloWorld.”
5. Select C# and type **HelloWorld.ascx** as File Name and **HelloWorld** as Module Name. You should have something like what is shown in [Figure 13.3](#).
6. Select Add Test Page and click Create Module. You should now have a screen like what is shown in [Figure 13.4](#).
7. Change to edit mode using the control bar's menu at the top right. You'll see the module gets its menu shown. Select Develop from the gear menu

([Figure 13.5](#)).

8. A pop-up screen appears with two tabs. On the first tab a drop-down with HelloWorld.aspx is selected. Delete all but the first line, type the following on the second line, and update and close the dialog:

```
<h1>Hello <%: UserInfo.DisplayName %></h1>
```

Hello SuperUser Account is displayed in the module. SuperUser Account is the `DisplayName` of the currently logged-in user. If you log off, you just see Hello, as there is no more user logged in.

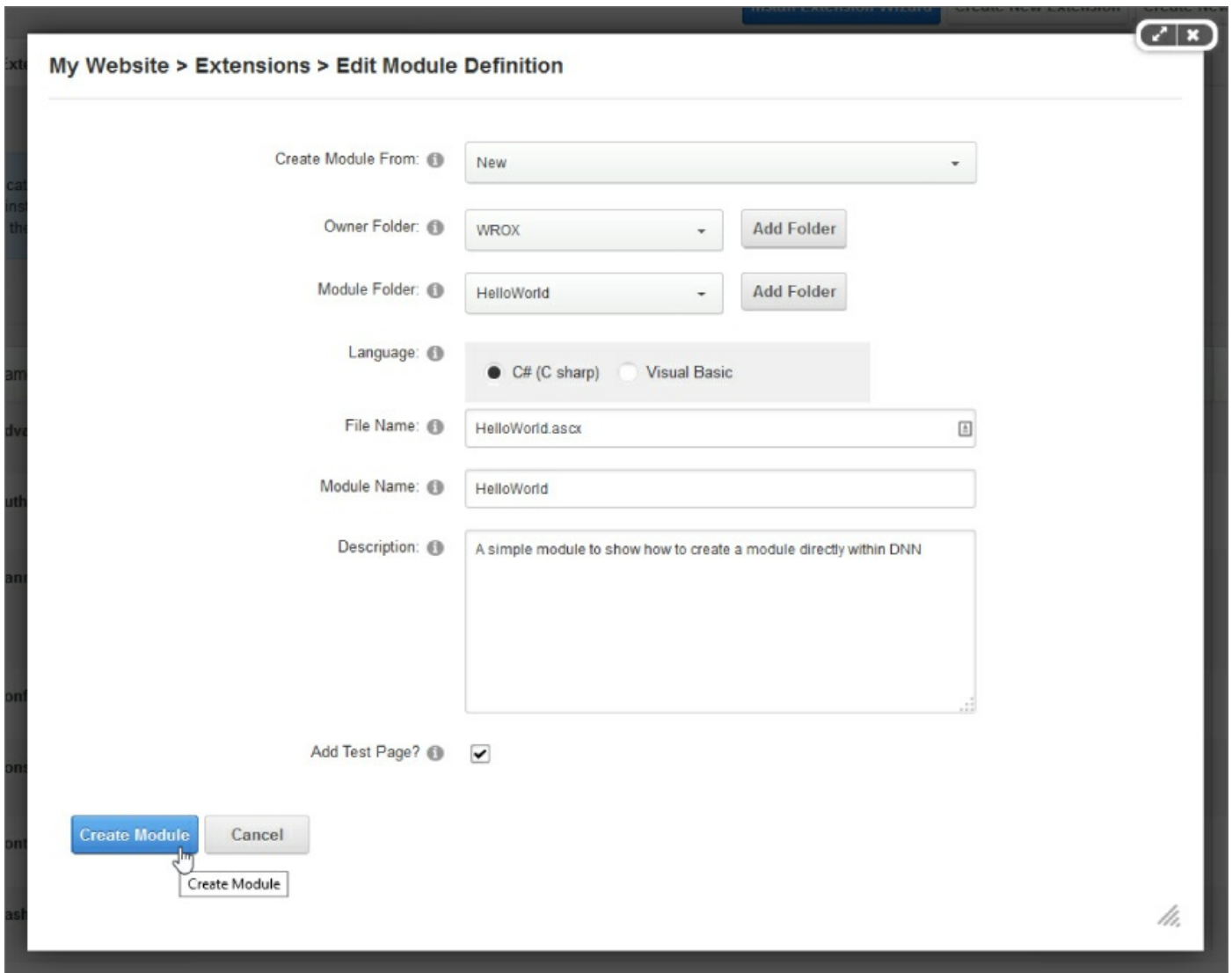


Figure 13.3

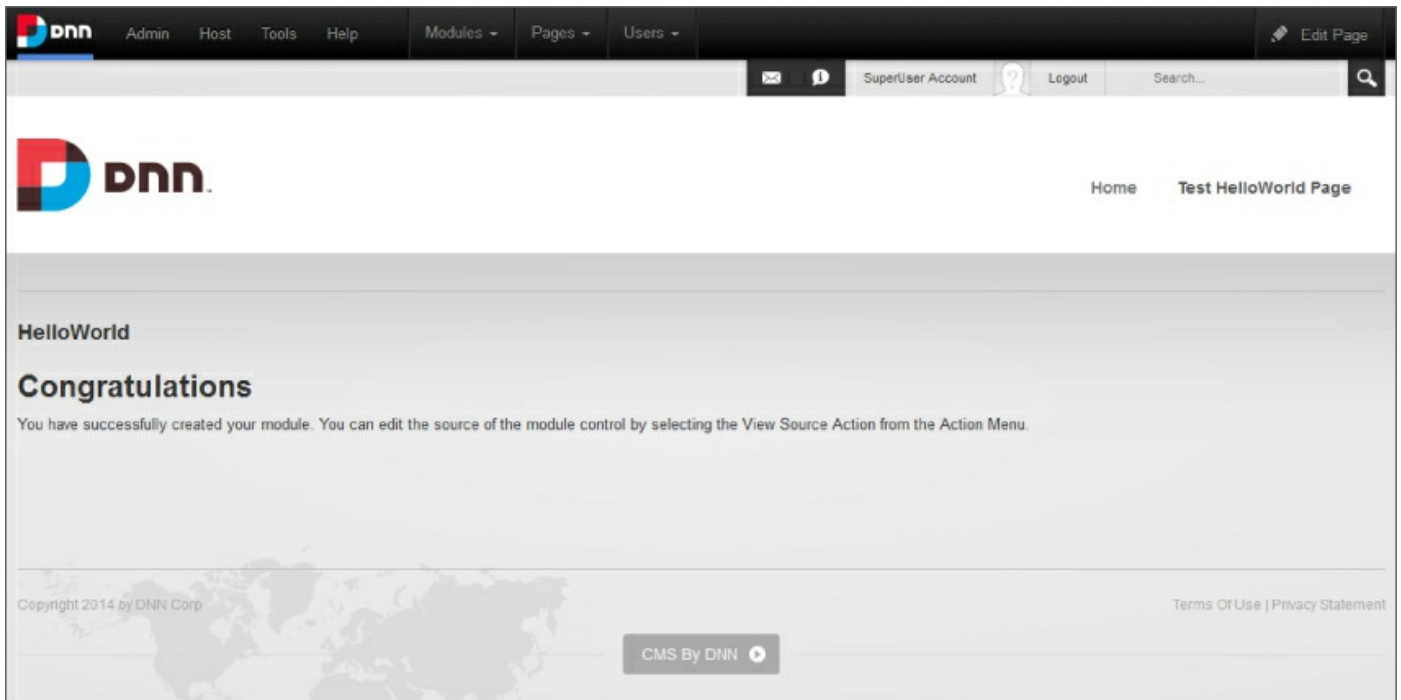


Figure 13.4

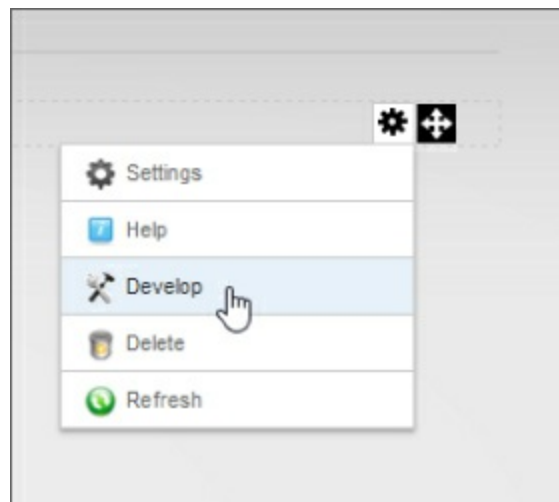


Figure 13.5

Congratulations. You've successfully created your first bit of active content! Now take it up a notch. Go back into development mode and copy the code in [Listing 13.1](#).

Listing 13.1: Hello World Web Forms example

```
<table>
<%
foreach (var tab in GetTabs()) {
%>
```

```

<tr>
  <td><%: tab.TabID %></td>
  <td>
    <a href="<%: DotNetNuke.Common.Globals.NavigateURL(tab.TabID)
%>">
      <%: tab.TabPath %>
    </a>
  </td>
</tr>
<%
}
%>
</table>

<script runat="server">
public List<DotNetNuke.Entities.Tabs.TabInfo> GetTabs()
{
  return
DotNetNuke.Entities.Tabs.TabController.GetPortalTabs(PortalId,
  -1, false, true);
}
</script>

```

What you should see is a list of pages with their path as a hyperlink to those pages in DNN. In the preceding listing there is a code block inside `script runat="server"` that runs solely on the server (you won't discover it in the HTML output of the page). This method is then used in the previous ASP.NET code to fill an HTML table. Finally, you'll note that you've used several methods from the DNN API (`NavigateURL`, `GetPortalTabs`). Finding your way around the DNN API takes some time, but the more modules you make, the more you'll find yourself using the same familiar bits of the framework.

The Razor Host Module

Over the years a lot of criticism has been leveled at Microsoft for its Web Forms implementation, in part because it was bulky and in part because Web Forms does not allow the programmer much control over the HTML that is emitted by the server. To address these issues we've seen the emergence of MVC and its associated view engine *Razor*. A Razor file is not unlike a Web Forms.ascx/.aspx in that it is basically a template telling the engine what HTML to render. Most significantly, you'll find that the `<% %>` way of separating code from markup has been replaced with a single `@` symbol. So `<%=Math.Sqrt(81)%>` is simply `@Math.Sqrt(81)`. But that is just a slight advantage in brevity. Razor files no longer support the Web Forms drag-and-

drop interface in Visual Studio. You need to craft the HTML yourself. Many, including us, see this as a step forward.

A lot of Razor's features are related to using MVC as a development approach. Although DNN will probably support that in the future, for now DNN still assumes you are rendering controls to a Web Forms page. To benefit from the Razor hype, DNN includes (since version 5.6.1) the so-called *Razor Host* module. It allows you to create a razor script, and the module renders this script on a DNN page. To access certain DNN features it includes three Helper Objects: `DnnHelper`, `HtmlHelper`, and `UrlHelper`. `DnnHelper` allows access to details about the current module, page, site, and user. The `HtmlHelper` makes accessing localized text easy. And finally the `UrlHelper` allows for easy construction of DNN URLs.

Use this feature to redo the Hello World example in the Razor Host module.

Hello World in Razor

Here we'll create a simple module using the Razor Host module that is included in DNN:

1. Create a new page and add the Razor Host module to this page.
2. From the module's pencil menu select Edit Script (make sure you're in Edit Mode if you don't see the menu).
3. Click to add a new script file and give it a meaningful name like HelloWorld. Click to add the file.
4. Select the file from the drop-down and replace the contents with this:

```
<h1>Hello @Dnn.User.DisplayName</h1>
```

5. Select the check box to make the script active and click to save and return to the page.

You now see what you had before—a welcome message for the logged-in user. Again, you add a list of pages in your current site with links to each of them (see [Listing 13.2](#)).

[Listing 13.2](#): Hello World Razor example

```
@functions {  
    public List<DotNetNuke.Entities.Tabs.TabInfo> GetTabs()  
}
```



```

    {
        return
DotNetNuke.Entities.Tabs.TabController.GetPortalTabs (Dnn.Portal.Port

        -1, false, true);
    }
}

<table>
@foreach (var tab in GetTabs())
{
    <tr>
        <td>@tab.TabID</td>
        <td>
            <a
href="@DotNetNuke.Common.Globals.NavigateURL (tab.TabID) ">@tab.TabPat

            </td>
        </tr>
    }
</table>

```

You can see the result in [Figure 13.6](#). As you can see from [Listing 13.2](#), Razor bears a lot of resemblance to Web Forms in the way it does the templating of your data. But there are some neat features that Web Forms doesn't have and vice versa. A fuller discussion of Razor falls outside the scope of this chapter, however.

```
Razor Host

Hello SuperUser Account

55 //Home
56 //Module
89 //Guestbook
88 //Razor
57 //ActivityFeed
60 //ActivityFeed//MyProfile
61 //ActivityFeed//Friends
62 //ActivityFeed//Messages
58 //Admin
63 //Admin//AdvancedSettings
64 //Admin//SiteSettings
65 //Admin//Pages
66 //Admin//Extensions
67 //Admin//Languages
68 //Admin//Skins
69 //Admin//SecurityRoles
70 //Admin//UserAccounts
71 //Admin//Vendors
72 //Admin//SiteLog
73 //Admin//Newsletters
75 //Admin//RecycleBin
76 //Admin//LogViewer
77 //Admin//SiteWizard
78 //Admin//Taxonomy
79 //Admin//SearchEngineSiteMap
80 //Admin//Lists
81 //Admin//SiteRedirectionManagement
82 //Admin//DevicePreviewManagement
83 //Admin//GoogleAnalytics
85 //Admin//FileManagement
59 //SearchResults
```

Figure 13.6

Final Remarks

Although it's perfectly valid to develop your module inline in DNN, you'll obviously miss out on the smoother experience of using a full IDE. Plus, you

won't be able to compile your code. It's fine for a half-day quickie, but for anything more elaborate you'll want to use Visual Studio. For the remainder of the chapter, we assume you are using Visual Studio to develop your module.

WAP versus WSP

Now you face one of the oldest dichotomies in module development. When DNN was first created, Visual Studio (2003) included a so-called Web Application Project (WAP) template. This template assumed that you were developing a number of user controls inside a larger Web Forms application and that you would compile the code behind to a DLL in the bin folder. That is exactly the route DNN took with modules. So to many it came as a nasty surprise that in Visual Studio 2005 this option was deprecated in favor of Web Site Projects (WSP), which took a very different approach. Microsoft later corrected this with a service pack, but it was an illustration of how you can be left behind by developments in Redmond.

In a WAP project you load your controls and refer to the rest of the project only by including the DLLs as references in your project. The result of your work will be the .ascx files plus a DLL that is the compiled code of your project. With WSP you load the entire website (DNN) and add code files to the App_Code folder that is then compiled dynamically when your application first loads. WSP was obviously meant to speed up development as you could change code and just click Refresh in your browser. But having template (.ascx) files under DesktopModules and the code files “far away” under App_Code was a change that was not conducive to larger, more complicated modules. Plus, it made creating a distributable module significantly more complex. And it also meant installing the source code on the client's server, which ruled out all commercial module developers.

To this day you can create a module using either WAP or WSP approach. But most developers use the WAP approach because it totally isolates your work from DNN, and that is seen as a bonus. The WSP approach actually makes sense only when faced with a quick-and-dirty task for a single site.

Inside versus Outside the Root

If you have downloaded and unpacked the source version of DNN, you may have noticed something odd: the modules are not under `\Website\DesktopModules\etc` where you might have expected to find them.

Instead, they are under `\Dnn Platform\Modules`. So how does this work? The way that the project has been set up is with the module projects outside the root of the website (that is, outside the `DesktopModules` folder). The site is running under `\Website`, so how do developers work on the module and see their changes? Inevitably this means that if you make a change in one of the `.ascx` files, it won't be visible when you refresh your browser.

The clue is in MSBuild tasks. What you need to do is to click Rebuild in Visual Studio. What happens is that some extra MSBuild tasks run at the end of the regular compilation of the module, which copies over all the files to the `Website\DesktopModules` folder. This is a very clever way of leveraging build automation. The upside of this way of organizing your project is that it is even more isolated from the website. And you can presume that for such a large project as the DNN Platform, this makes good sense. But it remains rare for module developers to use this approach. Plus, you don't work on the DNN site itself simultaneously. So you are not expected to interfere at all with the DNN files. That already provides enough isolation.

In short, it is possible to develop your module in total isolation in some directory far away from your test DNN site, but it requires quite a bit more jumping through hoops, and the benefits are limited. So in the remainder of the chapter, we assume you are developing inside the test DNN installation.

Module Design Considerations

Now that you have your bearings and you know what you want to accomplish with the module, let's look at some aspects that will have an influence on how you will code this. What follows are a number of decision points you will come across while you move from your module's concept to code.

Three-Tiered Design and MVC

By far the most common architecture in ASP.NET Web Forms is the three-tiered approach, where you have a presentation layer, a business layer, and a data access layer (DAL). This approach also pervades the DNN Platform. The idea is to separate your code between these three layers so that it becomes easier to make changes in one layer without affecting any others. This makes the solution easier to maintain. Concretely, the .ascx files and their code behind would constitute the presentation-layer code, the SQL methods of the data layer, and the various controllers and objects of the business layer. In DNN, you'll find the entire data layer under the namespace `DotNetNuke.Data`.

But as simple as it sounds, it's difficult to keep consistent, and Web Forms has been faulted by many for not encouraging developers enough to keep their code well separated. MVC is seen as a logical progression in web development, and it claims to provide a better “separation of concerns.” DNN 7.5 is slated to provide support for modules that wish to follow this programming paradigm. A full discussion of the merits of MVC falls outside the scope of this book, however, and given that support for MVC has not yet been implemented in DNN as of this writing, we focus on a three-tiered approach in the remainder of this chapter. Keep in mind that this is a totally valid way of creating your modules.

DAL

The data access layer is probably the easiest layer to recognize in any DNN module (and DNN itself). Again, there have been significant changes in the technical landscape since DNN was first created. The DAL is responsible for persisting the objects of your application to some data store. Or to put it more simply: It stores the various records in the various database tables of your application and can retrieve those again.

Because DNN was designed to be very flexible and extensible, many parts were designed following the *provider model pattern*. Basically, you have an

abstract class defining a number of abstract methods, and then you have a concrete implementation of the class, which is constructed by some factory method at runtime. This allows DNN to support databases other than SQL Server like MySQL or Oracle. If you want to run DNN on Oracle, all you need to do is create the concrete class that implements all the defined methods, and you need to create the various scripts to create the correct database schema in Oracle. This has been done in the past, and it was a commercial project. But it's an enormous undertaking with limited benefits. What turned out to be the biggest stumbling block is that it needed to be repeated for each and every module you had if you wanted to run your DNN site on Oracle. So more and more it is assumed that you are using DNN on SQL Server, especially because SQL Server has been accepted as an enterprise-grade database and because there are free entry-level versions of this. But you'll see the legacy of this in the naming of some parts, like the `.SqlDataProvider` filename extension for the scripts for SQL Server. As you can imagine, the Oracle provider had `.OracleDataProvider` files.

This development is why DNN has been moving away from the provider pattern for a while now. The first move was to add generic methods in the data provider that allowed module developers to avoid having to code the data provider abstract class and SQL data provider implementation in their own code for each and every function. Instead, you could code your method directly calling a specific stored procedure. This mechanism is referred to as DAL+.

There have been two developments in the data access domain since the original DNN that have had significant impact on ASP.NET in general and the way DNN interacts with data specifically: ORM and LINQ. To start with the latter, LINQ is a set of extensions to the .NET languages that allow developers to make SQL-like queries of various collection types (arrays, dictionaries, and so on). It has become hugely popular since its introduction in .NET 3.5. There are many good resources on LINQ, and we assume you are familiar with it. ORMs (Object Relational Mappers) remove the repetitive task for developers of creating CRUD (Create, Read, Update, Delete) statements for their objects between their code and the database. Instead, the developer can hand over an object and tell the ORM “just store this” and the ORM crafts the SQL dynamically to make this happen. ORMs, like Microsoft's Entity Framework, can be an incredibly powerful tool and time winner for developers. The most important downside to their use, however, is that (for all ORMs that have

been reviewed for inclusion in DNN) there is a performance penalty. So instead, DNN has chosen to adopt a so-called micro-ORM. A micro-ORM is not quite as powerful as a full ORM, but what you lose in flexibility you get back in performance.

DNN 7 ships with PetaPoco and introduces a new way of doing data access: DAL 2. This new layer promises to cut down the number of lines of code that module developers need to create significantly.

Client-Side versus Server-Side

When ASP.NET was first released in 2001, the thinking in the IT community was still very much thin client. Browsers were pretty dumb, buggy (IE), and nonstandardized (again IE), so the best approach a developer could take was to do as much as possible on the server and limit what the browser did to just displaying HTML and sending back form data. This worked for a number of years until it was discovered that with JavaScript you could alter the contents of a page and do similar interactions to the server as what the browser does when posting back, eliminating the need for page postbacks. Ajax was born, and soon every developer was asked to provide solutions that didn't require page refreshes anymore. Microsoft released its own version of Ajax in ASP.NET using so-called *update panels*. Skip a few years and you see the emergence of jQuery and JSON. These two technologies make it trivially easy for developers to send data back and forth to the server and alter what is onscreen. And it beats Microsoft's solution on two very important aspects: load and performance. Update panels send back the whole page to the server, and the server responds with bulky HTML. This means that there are a lot of bytes going back and forth. With the more modern approach, the developer decides selectively what is being sent back and forth, making this a much more efficient affair. Fast-forward a few more years, and Microsoft launches its Web API, which is intended to make developing services for data exchange between a web page and the server a lot easier. This unlocked the full potential of jQuery/JSON for ASP.NET developers.

DNN 7 includes enhanced support for jQuery (you'll find the libraries under Resources/Libraries), JSON (DNN ships with the most popular Newtonsoft.Json.dll library), and Web API. What you should think about at this stage of your project is how you divide the work your module does. What is best done on the server, and what will you do using services or even plainly in JavaScript? Browsers are much better behaved these days, and you are

expected to provide a smooth experience on both high-bandwidth as well as low-bandwidth (mobile) connections. Naturally you need to keep security in mind, too. Services can be exploited, and every door you open on the server for your application is one more target for a hacker.

Localization

For those who live outside the United States, it is all too familiar to see some new technological development only to find out that it is U.S. English only or that it can't be used in a multilingual setting. This is not ill will but more of a blind spot of many developers. "It works for me." For years one of us has been involved in a team (mostly from Europe) that tries to identify internationalization issues in DNN and offer solutions. This team meets regularly, and if there is one takeaway from these meetings, it's that this is hard to do right especially if it was not taken into account at the very conception. Besides the obvious translation of onscreen text, there are differences in how we write numbers and dates and even the direction we write. Making an application that behaves in the same way in all cultures is a daunting task. We urge you not to underestimate this aspect of your module.

Technically, we discern between two types of localization: static and content localization. The former deals with the texts and graphics that we ship with our work such as buttons (Submit, Cancel, and so on), field texts (First Name), help text, and so on. Static localization is done through resource files. These are XML files with the .resx extension you find in App_LocalResources folders everywhere. These files consist of key-value pairs where the value is the text in one specific language for that file. The U.S. English text is in the file that is simply named MyControl.ascx.resx that you create and supply with your work. Then, if it needs to be translated to French, all a translator needs to do is create a resource file called MyControl.ascx.fr-FR.resx (fr-FR stands for French as it's spoken in France as opposed to fr-CA, which is French as it's spoken in Canada) with the French translations in it. DNN checks the user's browser settings and personal preferences to determine which languages the user prefers and then attempts to find the best match for each text on display. This process includes so-called *fallbacks* where one could specify that if the user is German speaking from Austria but no text can be found in de-AT, then maybe the system should look for the German file for Germany (de-DE) before falling back on U.S. English. As you can probably understand, this can get incredibly complex very quickly. Not to mention, we also support

overrides per site and at the host level. To keep this from totally bringing DNN to its knees, a lot of caching is taking place.

DNN Software provides the .resx files for the entire DNN Platform for five languages besides English: Spanish, German, Italian, French, and Dutch. But with some simple tools you can create your own translation for DNN if you need to. You can use the built-in editor in DNN through the languages page, or you can use an external tool like the DNN Translator that you can find on CodePlex. Resource files are normally zipped up into *language packs*. An example is the aforementioned core language pack. But you can also have module-specific language packs. As a module developer you may need to ask a translator to translate your resource files as a service for your customers so you can deliver these packs yourself.

For the most part, your concern with static localization means you use the `ResourceKey` attribute on your controls and add the key to the corresponding .resx file. There should be a big red warning light on your monitor that flashes every time you are hard-coding any bit of text to display to an end user. It just isn't done.

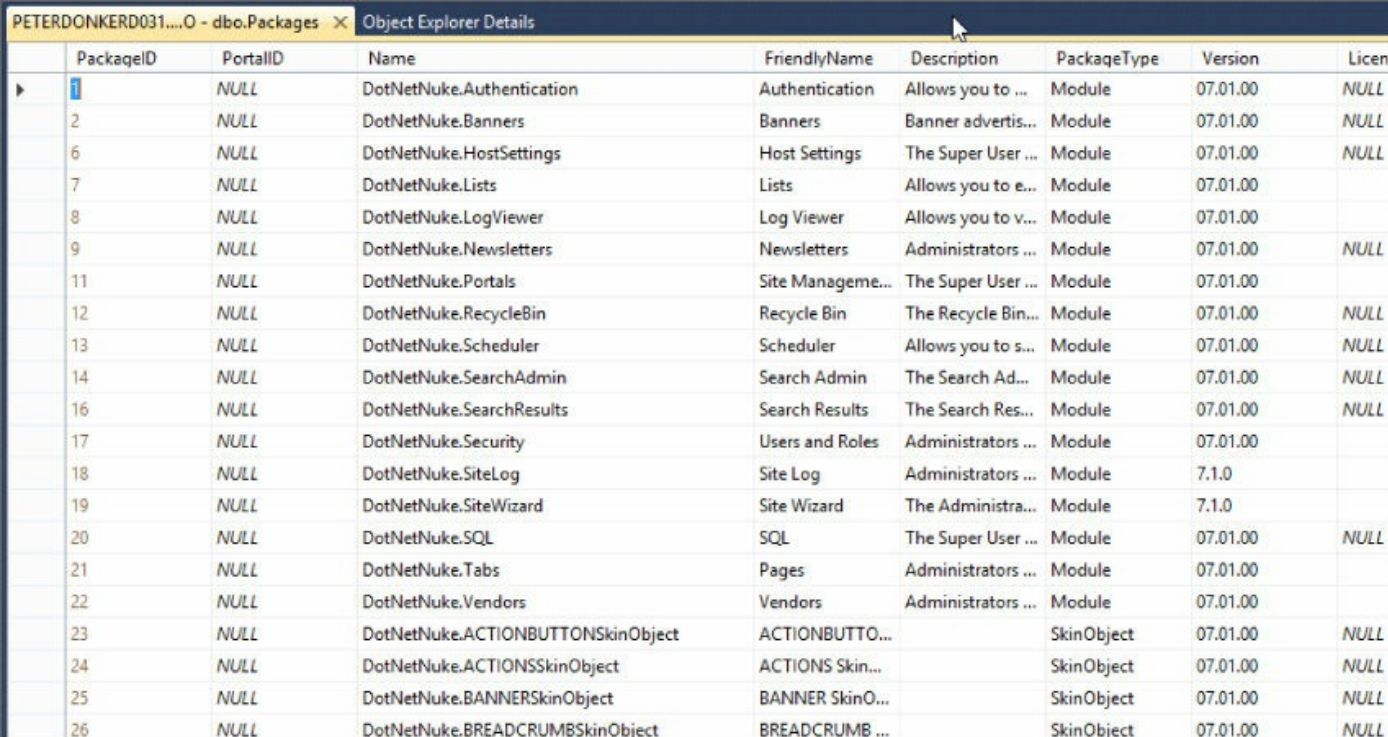
Content localization is the domain of multilingual sites. It concerns all the features built in to DNN to allow users to create content in different languages and have the site adapt to the user's preferences. The way it is implemented in DNN is that when a page has been designated to be multilingual, it in fact creates shadow copies of itself for the other languages that the site wants to support. It then copies all modules from the original page to those shadow pages. At this point, translators can work on those modules translating their content, and when done, they mark a page as done, and it becomes available in the site. Sometimes you might not want to have a copy of a module, but rather the same one. That is also possible in DNN. You can manage that on the languages interface for a page. What you should be aware of as a module developer is the mechanism just described. Depending on your application, you may (a) totally ignore the whole content localization aspect, (b) decide that for other languages the module should be copied as discrete instances, or (c) decide that your module should not be exploded into n versions but that the same module should appear on each page. In the latter case, we'd speak of a module that implements content localization within itself. In the second case, we let DNN handle content localization.

About Modules, TabModules, Module Definitions

Your module needs to be properly embedded in DNN for it to work. A surprising number of concepts come into play to make this happen. The best way to understand how this works is to look at what is stored in SQL. After all, that is where the registration of a module is stored. To get an idea of how the module is embedded in DNN, we open the following tables in turn and try to walk you through the wiring. We start at the very top where your module has been announced to the system and drill down to find out how DNN knows which .ascx to put where on the page.

Packages

Packages are what you upload into DNN: the Zip files. They contain the manifest, and this table stores this manifest, the license (if it was included), owner details, and so on. Note the column `PackageType` (Figure 13.7). You'll see the various package types that are known in DNN. For the purpose of this chapter, we are interested only in *Module* package types. `Packages` is the entry point of all extensions, and for our purpose it's the least interesting table.



PackageID	PortalID	Name	FriendlyName	Description	PackageType	Version	Licen
1	NULL	DotNetNuke.Authentication	Authentication	Allows you to ...	Module	07.01.00	NULL
2	NULL	DotNetNuke.Banners	Banners	Banner advertis...	Module	07.01.00	NULL
6	NULL	DotNetNuke.HostSettings	Host Settings	The Super User ...	Module	07.01.00	NULL
7	NULL	DotNetNuke.Lists	Lists	Allows you to e...	Module	07.01.00	
8	NULL	DotNetNuke.LogViewer	Log Viewer	Allows you to v...	Module	07.01.00	
9	NULL	DotNetNuke.Newsletters	Newsletters	Administrators ...	Module	07.01.00	NULL
11	NULL	DotNetNuke.Portals	Site Managemen...	The Super User ...	Module	07.01.00	
12	NULL	DotNetNuke.RecycleBin	Recycle Bin	The Recycle Bin...	Module	07.01.00	NULL
13	NULL	DotNetNuke.Scheduler	Scheduler	Allows you to s...	Module	07.01.00	NULL
14	NULL	DotNetNuke.SearchAdmin	Search Admin	The Search Ad...	Module	07.01.00	NULL
16	NULL	DotNetNuke.SearchResults	Search Results	The Search Res...	Module	07.01.00	NULL
17	NULL	DotNetNuke.Security	Users and Roles	Administrators ...	Module	07.01.00	
18	NULL	DotNetNuke.SiteLog	Site Log	Administrators ...	Module	7.1.0	
19	NULL	DotNetNuke.SiteWizard	Site Wizard	The Administra...	Module	7.1.0	
20	NULL	DotNetNuke.SQL	SQL	The Super User ...	Module	07.01.00	NULL
21	NULL	DotNetNuke.Tabs	Pages	Administrators ...	Module	07.01.00	
22	NULL	DotNetNuke.Vendors	Vendors	Administrators ...	Module	07.01.00	
23	NULL	DotNetNuke.ACTIONBUTTONSkinObject	ACTIONBUTTO...		SkinObject	07.01.00	NULL
24	NULL	DotNetNuke.ACTIONSSkinObject	ACTIONS Skin...		SkinObject	07.01.00	NULL
25	NULL	DotNetNuke.BANNERSkinObject	BANNER SkinO...		SkinObject	07.01.00	NULL
26	NULL	DotNetNuke.BREADCRUMBSkinObject	BREADCRUMB ...		SkinObject	07.01.00	NULL

Figure 13.7

DesktopModules

This is the data root for any module. Every module has a unique record in this table ([Figure 13.8](#)). You'll notice it refers back to `Packages` through `PackageId` and that the version is duplicated in this row (it was also present in the `Packages` table). This is because the `Packages` table is a recent addition. Most importantly this table tells DNN where the module is found (`FolderName`) and what it's called (`ModuleName`). Note that you should keep the latter unique to avoid clashes. This is because DNN uses this name to determine during installation whether it has already been installed. So choosing a name that is the same as an existing one will lead to a huge mess when people install your module and the other one is present as well. So stick to some convention like `MyCompany_MyModule` for the name. You won't see it in the UI as `FriendlyName` is used for that. Other aspects of note: `IsPremium` tells DNN whether it is only reserved for some sites or whether all sites can use this. `IsAdmin` tells DNN whether it should be available only for administrators to instantiate and use.

	DesktopModul...	FriendlyName	Version	IsPremium	IsAdmin	FolderName	ModuleName	SupportedFeatures	PackageID
▶	10	Users And Roles	07.01.00	True	False	Admin/Security	Security	0	17
	11	Pages	07.01.00	False	False	Admin/Tabs	Tabs	0	21
	14	Vendors	07.01.00	False	False	Admin/Vendors	Vendors	0	22
	15	Banners	07.01.00	False	False	Admin/Banners	Banners	0	2
	18	Site Log	07.01.00	False	False	Admin/SiteLog	SiteLog	0	18
	19	Newsletters	07.01.00	False	False	Admin/Newslet...	Newsletters	0	9
	22	Portals	07.01.00	True	True	Admin/Portals	Portals	0	11
	24	SQL	07.01.00	True	True	Admin/SQL	SQL	0	20
	30	Host Settings	07.01.00	True	True	Admin/HostSet...	HostSettings	0	6
	34	Recycle Bin	07.01.00	False	False	Admin/Recycle...	RecycleBin	0	12
	37	Log Viewer	07.01.00	False	False	Admin/LogVie...	LogViewer	0	8
	38	Scheduler	07.01.00	True	True	Admin/Schedul...	Scheduler	0	13
	43	Search Admin	07.01.00	True	True	Admin/Search...	SearchAdmin	0	14
	45	Search Results	07.01.00	False	False	Admin/SearchR...	SearchResults	0	16
	46	Site Wizard	07.01.00	False	False	Admin/SiteWiz...	SiteWizard	0	19
	48	Lists	07.01.00	False	False	Admin/Lists	Lists	0	7
	49	Account Login	07.01.00	False	False	Admin/Authen...	Authentication	0	1
	52	Extensions	07.01.00	True	True	Admin/Extensi...	Extensions	0	52
	56	Dashboard	07.01.00	True	True	Admin/Dashbo...	Dashboard	0	61
	57	Languages	07.01.00	False	False	Admin/Langua...	Languages	0	62
	58	Skins	07.01.00	False	False	Admin/Skins	Skins	0	63
	59	Site Settings	07.01.00	True	True	Admin/SiteDe...	SiteSettings	0	64

Figure 13.8

ModuleDefinitions

Next up is the module definition ([Figure 13.9](#)). It links to the `DesktopModules` table. So any module can contain one or more definitions. Each definition shows up as a component on the page when you instantiate the module (that is, drop it on the page). Most modules have just one definition. Multiple

definitions falls outside the scope of this chapter (and we're not big fans of it to be honest). The Module Definition has a `DefinitionName`. Like the `ModuleName` this is best kept unique and stable. If you change the name of this, DNN creates a new definition next time you upgrade, and then you have a module with multiple definitions and hence multiple components onscreen (probably showing the same thing). It can get very messy if you start fidgeting with this value, so choose it well right at the start along the same conventions mentioned previously.

	ModuleDefID	FriendlyName	DesktopModul...	DefaultCacheT...	DefinitionName
▶	12	Security Roles	10	0	Security Roles
	13	Tabs	11	0	Tabs
	14	Site Settings	22	0	Site Settings
	15	User Accounts	10	0	User Accounts
	19	Vendors	14	0	Vendors
	20	Banners	15	0	Banners
	27	Site Log	18	0	Site Log
	28	Newsletters	19	0	Newsletters
	63	Portals	22	0	Portals
	65	SQL	24	0	SQL
	72	Host Settings	30	0	Host Settings
	74	Account Login	49	-1	Account Login
	75	User Account	10	0	User Account
	76	Recycle Bin	34	0	Recycle Bin
	79	Log Viewer	37	0	Log Viewer
	80	Scheduler	38	0	Scheduler
	85	Search Admin	43	0	Search Admin
	87	Search Results	45	0	Search Results
	88	Site Wizard	46	0	Site Wizard
	90	Lists	48	0	Lists
	94	Extensions	52	0	Extensions
	98	Dashboard	56	0	Dashboard

Figure 13.9

ModuleControls

The module definition is basically a container for one or more module controls. The module control ([Figure 13.10](#)) tells DNN which `.ascx` to show (yes, we're finally getting to your work) when the module is shown on a page. The `ControlSrc` is the relative path to the `.ascx` to show. This `.ascx` has to either inherit from `DotNetNuke.Entities.Modules.PortalModuleBase` or implement `DotNetNuke.UI.Modules.IModuleControl`. You can actually use a type reference in this field if you want to. So how does DNN know which

module control to load for a definition? That depends on the `ControlKey`. This key is used in the query string of the request to switch controls. So adding `ctl=Edit` would tell DNN to load the control where `ControlKey` equals `Edit`.

	ModuleControlID	ModuleDefID	ControlKey	ControlTitle	ControlSrc	ControlType
▶	19	12	NULL	NULL	DesktopModules/Admin/Security/Roles.ascx	0
	20	12	Edit	Edit Security Ro...	DesktopModules/Admin/Security/EditRoles.ascx	1
	21	13	NULL	NULL	DesktopModules/Admin/Tabs/Tabs.ascx	0
	22	13	Edit	Edit Tabs	DesktopModules/Admin/Tabs/ManageTabs.ascx	0
	23	14	NULL	NULL	DesktopModules/Admin/Portals/SiteSettings.ascx	0
	24	15	NULL	NULL	DesktopModules/Admin/Security/Users.ascx	0
	25	15	Edit	Edit User Accou...	DesktopModules/Admin/Security/ManageUsers.ascx	1
	26	19	NULL	NULL	DesktopModules/Admin/Vendors/Vendors.ascx	0
	27	19	Edit	Edit Vendors	DesktopModules/Admin/Vendors/EditVendors.ascx	1
	28	20	NULL	NULL	DesktopModules/Admin/Banners/DisplayBanners.ascx	0
	29	20	Edit	Edit Banners	DesktopModules/Admin/Banners/BannerOptions.ascx	1
	34	27	NULL	NULL	DesktopModules/Admin/SiteLog/SiteLog.ascx	0
	35	28	NULL	NULL	DesktopModules/Admin/Newsletters/Newsletter.ascx	0
	41	19	Banner	Banner	DesktopModules/Admin/Vendors/EditBanner.ascx	1
	42	12	User Roles	User Roles	DesktopModules/Admin/Security/SecurityRoles.ascx	1
	51	63	Signup	Signup	DesktopModules/Admin/Portals/Signup.ascx	-1
	52	63	NULL	NULL	DesktopModules/Admin/Portals/Portals.ascx	0
	53	63	Edit	Edit Portals	DesktopModules/Admin/Portals/SiteSettings.ascx	3
	56	65	NULL	NULL	DesktopModules/Admin/SQL/SQL.ascx	0
	67	72	NULL	NULL	DesktopModules/Admin/HostSettings/HostSettings.ascx	0
	71	74	NULL	NULL	DesktopModules/Admin/Authentication/Login.ascx	-1
	73	75	NULL	User Account	DesktopModules/Admin/Security/ManageUsers.ascx	-1

Figure 13.10

That has brought us to your control. We now know how DNN knows which `.ascx` to load based on your module's registration in the framework. You now see what parts are involved when a module is instantiated on a page. Again, we walk through the data to get an idea of how this is done.

Modules

This is the root of every module in your site. Every module instantiation has a single record in this table ([Figure 13.11](#)). Note it does not tell DNN yet where it is placed on your site, just that it exists (or is deleted, and so on). Most importantly this is where the `ModuleID` is generated. That is the unique key that you use throughout your code to keep stuff from one module separate from another.

	ModuleID	ModuleDefID	AllTabs	IsDeleted	InheritViewPermissions	StartDate	EndDate	PortalID
	350	98	False	False	True	NULL	NULL	NULL
	352	102	False	False	True	NULL	NULL	NULL
	354	108	False	False	True	NULL	NULL	NULL
	362	117	False	False	True	NULL	NULL	0
	363	117	False	False	True	NULL	NULL	0
	364	117	False	False	True	NULL	NULL	0
	365	117	False	False	True	NULL	NULL	0
	366	117	False	False	True	NULL	NULL	0
	367	117	False	False	True	NULL	NULL	0
	368	113	False	False	True	NULL	NULL	0
	369	113	False	False	True	NULL	NULL	0
	370	123	False	False	True	NULL	NULL	0
	371	123	False	False	True	NULL	NULL	0
	372	117	False	False	True	NULL	NULL	0
	373	117	False	False	True	NULL	NULL	0
	374	105	False	False	True	NULL	NULL	0
	375	119	False	False	True	NULL	NULL	0

Figure 13.11

TabModules

The TabModule is where the module meets the page (Tab). It tells DNN where the module should go. As you can understand, you can have multiple TabModule per module. That is, you can stick a module on multiple pages. The idea behind this is that you can share content across pages. To see this in action, you can try the following: Open your DNN, log in as admin, and go to the Modules menu at the top. Instead of selecting Add New Module, select Add Existing Module. You now need to select a page from where to copy the module from and select the module. If you do not select Make a Copy, only a new TabModule record is created for that module, so it shows exactly the same content as on the other page.

You'll notice in the TabModules table ([Figure 13.12](#)) the columns PaneName and ModuleOrder. This tells DNN where to stick it on the current page given the skin you used when you added the module. This explains why when you switch skins, your modules are sometimes all stacked in the default pane. This is because DNN tries to find the pane on the selected skin, but if it isn't found (because the other skin had different pane names), the module is added to the (mandatory) ContentPane.

	TabModuleID	TabID	ModuleID	PaneName	ModuleOrder	CacheTime	Alignment	Color	Border
▶	41	7	352	ContentPane	1	0	NULL	NULL	NULL
	17	16	327	ContentPane	1	0	NULL		
	12	17	321	ContentPane	3	0	NULL	NULL	NULL
	15	20	324	ContentPane	9	0	NULL		
	16	21	325	ContentPane	11	0	NULL		
	21	25	332	ContentPane	1	0	NULL	NULL	NULL
	31	33	342	ContentPane	1	0	NULL	NULL	NULL
	32	34	343	ContentPane	1	0	NULL	NULL	NULL
	34	36	345	ContentPane	1	0	NULL	NULL	NULL
	39	37	350	ContentPane	1	0	NULL	NULL	NULL
	43	40	354	ContentPane	1	0	NULL	NULL	NULL
	53	55	364	contentpaneLo...	1	1200	NULL	NULL	NULL
	55	55	366	footerCenterPa...	1	1200	NULL	NULL	NULL
	52	55	363	footerLeftOuter...	1	1200	NULL	NULL	NULL
	54	55	365	footerLeftPane	1	1200	NULL	NULL	NULL
	56	55	367	footerRightOut...	1	1200	NULL	NULL	NULL
	51	55	362	footerRightPane	1	1200	NULL	NULL	NULL
	59	55	370	leftPane	1	0	NULL	NULL	NULL
	60	55	371	leftPane	2	0	NULL	NULL	NULL

Figure 13.12

There are many more columns in this table that we will not elaborate on here. You'll probably notice they are closely linked to settings you see when you go to a module's settings pop-up. But now you should understand how DNN knows which control to stick where when it loads a page.

A Guestbook Module

Let's take a nontrivial example to build a module. We're going to build a guestbook module. The module allows users to leave a message, and these messages are displayed in a list that is sorted by the date they were added in reverse order. There is a setting in the module specifying whether we wish to use moderation. If not, then all messages are displayed as they come in. If we use moderation, then anyone with edit permissions on the module needs to click a button to OK any incoming message before it's shown. Finally, users can edit their own messages until they've been moderated, and any user with edit permissions can edit any message. If a message has been edited, it displays with a small note about who edited the message and when.

The tools we're using are

- Visual Studio 2013
- Community Build Tasks
- Christopher Hammond's DNN Module Templates
- DNN 7.3.4

Module Templates and Build Tasks

As you'll quickly discover, making DNN modules involves a number of repetitive tasks. We wouldn't be software engineers if we didn't come up with a way to automate this. Two things we recommend right from the start are to use a Visual Studio project template and to install the community build tasks. Both are products of individual members of the DNN community. There is no official release of these by DNN Corp, and there are several competing versions of both. For this demo we use Christopher Hammond's module template, as he has a long track record in this. It is certainly a great template to begin with. Another one is Bitboxx's DNN 7 module template that focuses on using the new DAL and includes T4 templates to generate code from the database. Once you're familiar with module programming, we urge you to explore and adapt a template to suit your own needs. Editing templates for Visual Studio is fairly trivial, and quite quickly you'll find yourself wanting to tweak them. In our example we've already edited Chris's template to use WROX as company name, for example.

Most templates rely on community build tasks being installed. Build tasks are

MSBuild instructions that a project can leverage during the build process. Specifically for DNN it is useful to automate the creation of a module package. This is why these tasks have become quite popular by DNN module developers. Again, there are competing products out there. The oldest is maintained by Vicenç Masanas and is called MSBuild DotNetNuke Tasks. We have those installed. But there is a newer project called MS Build for DNN module development that is maintained by Ernst-Peter Tamminga. They are both respected long-serving members of the DNN community, so we're not going to push one or the other on you.

Creating the Project

We assume you've already installed a DNN install version and you've installed the tasks and module template.

1. Start Visual Studio and select New Project.
2. Select the DotNetNuke 7 C# DAL2 Compiled Module template from the Visual C#/DotNetNuke folder. Set the name to Guestbook and the folder path to wherever your DNN installation is and then DesktopModules\WROX\. Make sure to deselect the Create directory for solution check box before you click OK ([Figure 13.13](#)).

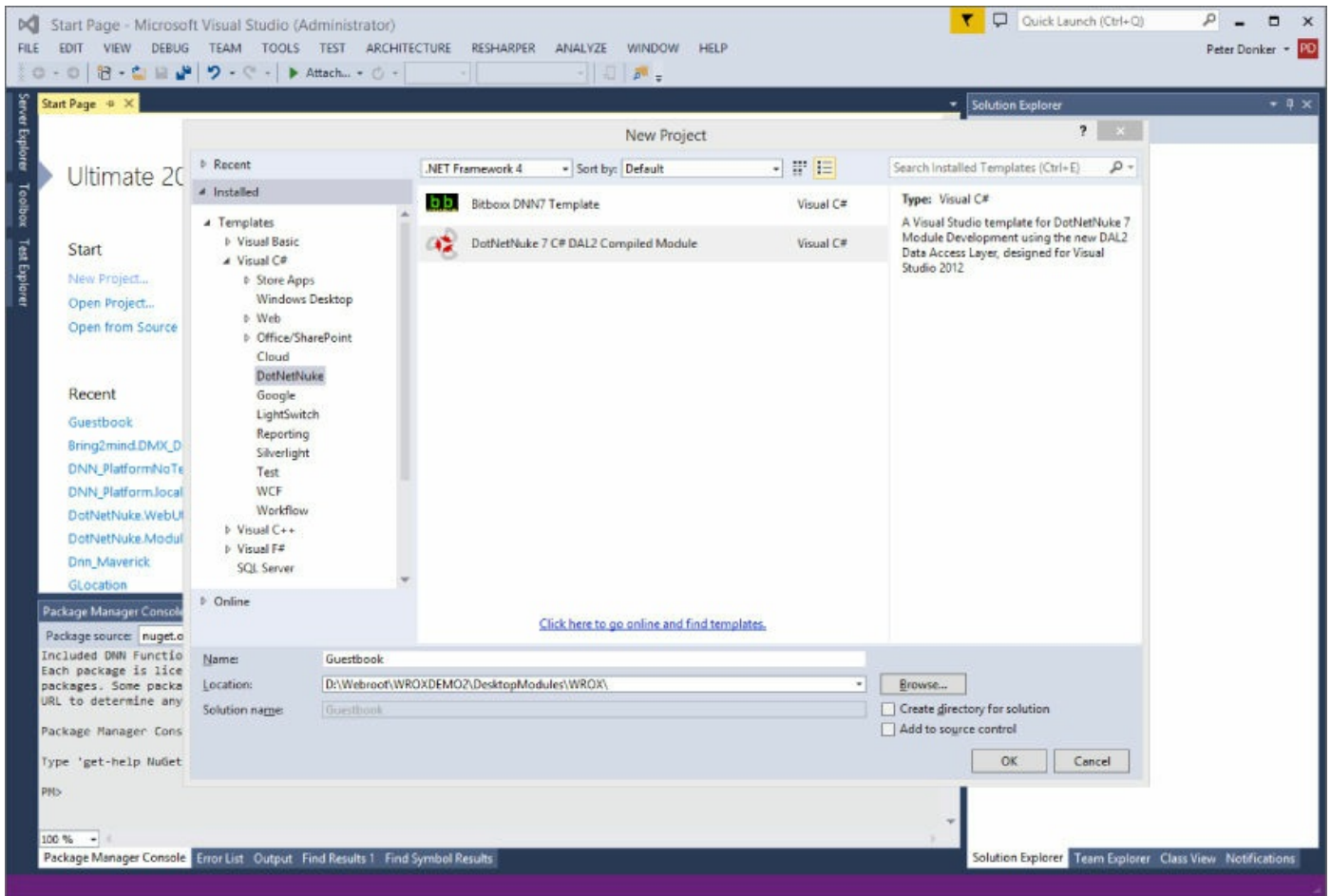


Figure 13.13

You should now see the new project loaded in your Visual Studio ([Figure 13.14](#)).

1. Right-click References and browse to add references to DotNetNuke.dll and DotNetNuke.WebUtility.dll from the bin folder of the site in which you're developing. The list of references should no longer show any errors.
2. Open Guestbook.dnn. This is the module manifest. The manifest needs to be packed with a module when you distribute it, and it tells DNN how to load the module. You look more in depth into the manifest in the next section. For now, you're going to edit the manifest so you can initialize your module in DNN. You'll be focusing your attention to the segment that starts with

```
<component type="Module">
```

3. Edit or verify the manifest so the following lines are as they are here:

```
<foldername>WROX\Guestbook</foldername>
```

```
...
```

```
<controlSrc>DesktopModules/WROX/Guestbook/View.ascx</controlSrc>
...
<controlSrc>DesktopModules/WROX/Guestbook/Edit.ascx</controlSrc>
...
<controlSrc>DesktopModules/WROX/Guestbook/Settings.ascx</controlSrc>
```

4. Fire up your dev site, log in as host, and go to the Extensions page.
5. Click Create New Module and select to create the module from a manifest. Select the right folder name (WROX) and module folder (Guestbook). In the resource drop-down you should see your .dnn file that you just edited. Select Add Test Page and click to create the module ([Figure 13.15](#)).

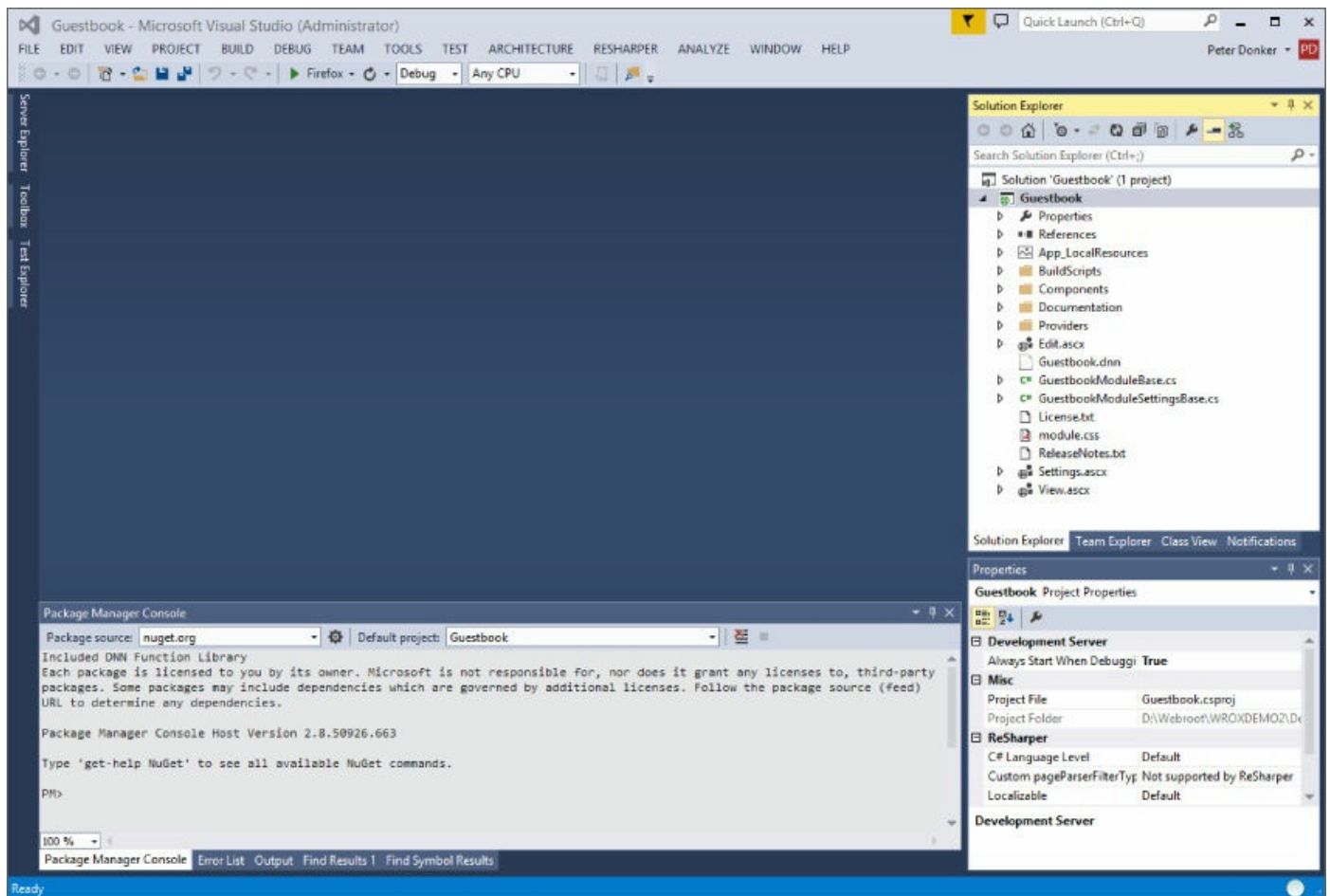


Figure 13.14

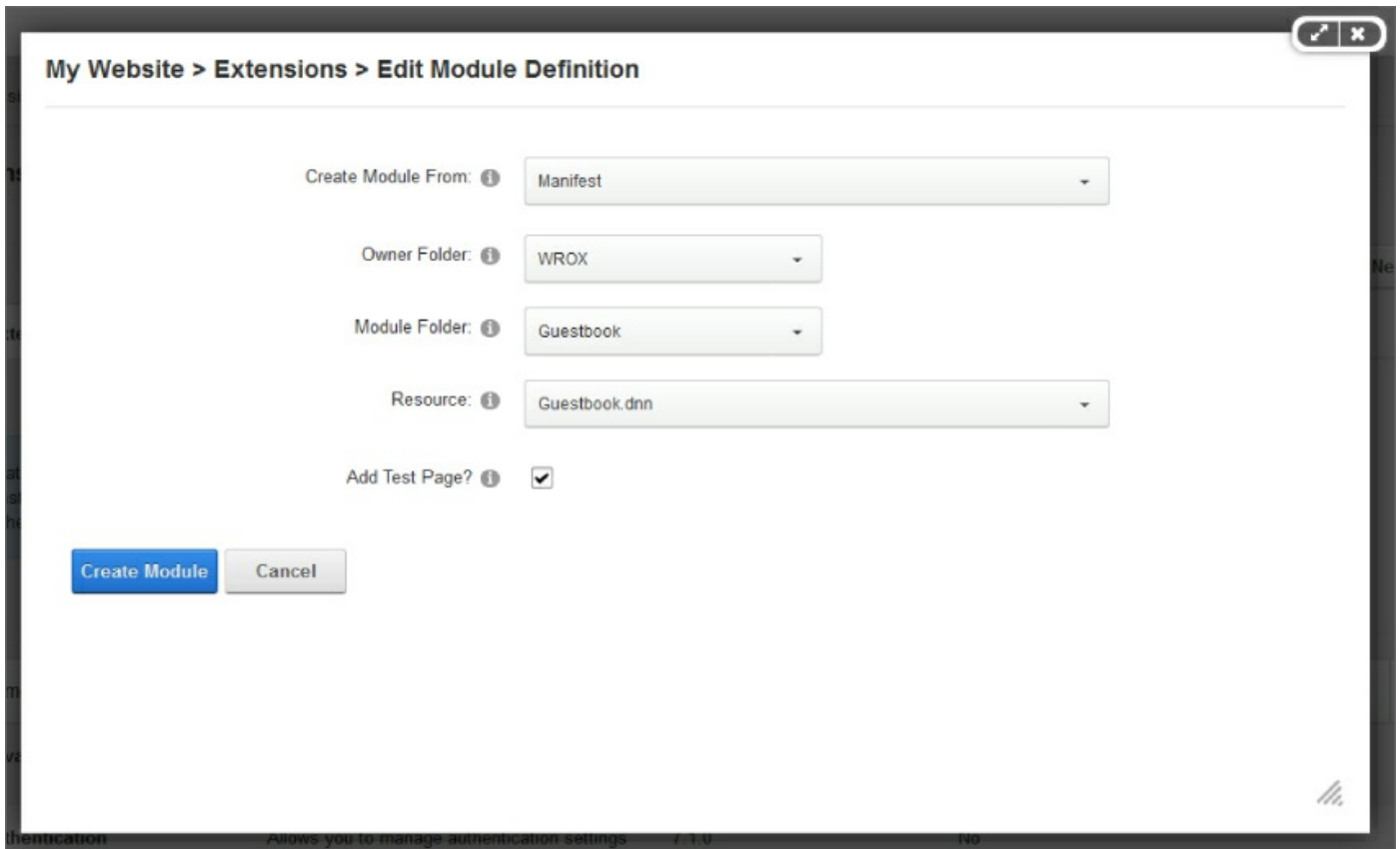


Figure 13.15

You are now redirected to the test page, and you'll see a big error. That is because you haven't actually coded anything yet. All you've done is create a new module definition in DNN using the controls listed in the manifest. Your files are still filled with boilerplate stuff, and there is no DLL in the bin folder with your code. For now you can close the browser as we will flesh out the module.

The Data Layer—SQL

It's time to think about what it is you're going to be storing. Foremost, you'll be storing a text message left by visitors. Call the object a guestbook entry. The entry will have message as field as well as a date time stamp when it was created and a link to the users table in DNN to store who it was that left the message. You also need to bring in scope, meaning you need to record the module ID in which the entry was created. This avoids the message cropping up in another module if the user adds multiple guestbook modules to his site. Finally, you need a flag to tell whether the message has been approved. The SQL to create this is shown in [Listing 13.3](#).

Listing 13.3: SQL script to create entries table

```
CREATE TABLE {databaseOwner}
{objectQualifier}WROX_Guestbook_Entries (
  [EntryId] [int] IDENTITY(1,1) NOT NULL,
  [ModuleId] [int] NOT NULL,
  [Message] [nvarchar](max) NOT NULL,
  [Approved] [bit] NULL,
  [CreatedByUserID] [int] NULL,
  [CreatedOnDate] [datetime] NULL,
  [LastModifiedByUserID] [int] NULL,
  [LastModifiedOnDate] [datetime] NULL,
  CONSTRAINT PK_{objectQualifier}WROX_Guestbook_Entries PRIMARY KEY
(EntryId)
)
GO

ALTER TABLE {databaseOwner}{objectQualifier}WROX_Guestbook_Entries
ADD CONSTRAINT FK_{objectQualifier}WROX_Guestbook_Entries_Modules
FOREIGN KEY([ModuleId])
REFERENCES {databaseOwner}{objectQualifier}Modules (ModuleID)
ON DELETE CASCADE
GO
```

You'll notice we've used two tokens that are specific to DNN and will throw errors if you try to run this as is in SQL Management Studio. If you run this in the SQL module (Host menu), then it works just fine. If you want to run it directly in SQL Management Studio, then the {databaseOwner} should be replaced with "dbo" most likely (the default db owner) and {objectQualifier} with nothing as by default DNN does not add this. The object qualifier is a remnant of DNN's long history and is discussed later in this book.

Another thing you'll notice is that we've prefixed our table with WROX_Guestbook_. It is good practice to prefix your SQL objects with a unique string so there is little chance of it colliding with another third-party module. Every module adds its data to SQL, and it all has to live peacefully side by side, so a little precaution in naming items comes in handy.

Finally, we've added the standard audit columns to the table (Created/LastModified). You'll find these in many places in DNN's data layer, and we find it good practice to stick to conventions when they correspond to your situation. Here we'll keep a record of who created the message and the last one to have modified it (remember, there is an option to edit the

message).

We also create a view to correspond to our table that brings in the display names of the creator and modifier of an entry. This view is created as shown in [Listing 13.4](#).

[Listing 13.4](#): SQL script to create entries view

```
CREATE VIEW {databaseOwner}
{objectQualifier}vw_WROX_Guestbook_Entries
AS
SELECT
    e.*,
    ISNULL(uc.DisplayName, 'Unknown') AS CreatedByUserDisplayName,
    ISNULL(um.DisplayName, 'Unknown') AS
    LastModifiedByUserDisplayName
FROM {databaseOwner}{objectQualifier}WROX_Guestbook_Entries e
    LEFT JOIN {databaseOwner}{objectQualifier}Users uc
        ON e.CreatedByUserID = uc.UserID
    LEFT JOIN {databaseOwner}{objectQualifier}Users um
        ON e.LastModifiedByUserID = um.UserID
GO
```

As you can see, we're doing a left joins for these users as we can imagine a scenario where this is an anonymous post, in which case we'll just use the name `Unknown` for that user.

This is all we'll be doing in SQL Server. We now have a table and a view for our data. We can now turn to Visual Studio and code up our module.

The Data Layer—DAL 2

Let's glue the SQL to code using a DAL 2 approach. We begin by deleting stuff we don't need from our newly created module. So delete all files in the Components folder, and you can delete the entire `Documentation` folder as well. Now add a new file to the Components folder called `EntryInfo.cs`. The `EntryInfo` object will be what is mapped to our table. We annotate the class to give it meaning in DAL 2 as shown in [Listing 13.5](#).

[Listing 13.5](#): EntryInfo.cs

```
using System;
```

```

using DotNetNuke.ComponentModel.DataAnnotations;

namespace WROX.Modules.Guestbook.Components
{
    [Scope("ModuleId")]
    [TableName("WROX_Guestbook_Entries")]
    [PrimaryKey("EntryId")]
    public class EntryInfo
    {
        public int EntryId { get; set; }
        public int ModuleId { get; set; }
        public string Message { get; set; }
        public bool Approved { get; set; }
        public int CreatedByUserID { get; set; }
        public DateTime CreatedOnDate { get; set; }
        public int LastModifiedByUserID { get; set; }
        public DateTime LastModifiedOnDate { get; set; }
    }
}

```

Most of this listing is self-explanatory. The scope attribute tells DAL 2 that we'll be passing in `ModuleId` to limit what we access.

Next we'll add the controller class that will do our CRUD operations. Create a file called `EntryController.cs` in the same folder as shown in [Listing 13.6](#).

Listing 13.6: EntryController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using DotNetNuke.Data;

namespace WROX.Modules.Guestbook.Components
{
    public class EntryController
    {
        public static void AddEntry(EntryInfo entry, int userId)
        {
            entry.CreatedByUserID = userId;
            entry.CreatedOnDate = DateTime.Now;
            entry.LastModifiedByUserID = userId;
            entry.LastModifiedOnDate = DateTime.Now;
            using (IDataContext ctx = DataContext.Instance())
            {
                var rep = ctx.GetRepository<EntryInfo>();
                rep.Insert(entry);
            }
        }
    }
}

```

```
    }  
  }  
  
  public static EntryInfo GetEntry(int entryId, int moduleId)  
  {  
    EntryInfo entry;  
    using (IDataContext ctx = DataContext.Instance())  
    {  
      var rep = ctx.GetRepository<EntryInfo>();  
      entry = rep.GetById(entryId, moduleId);  
    }  
    return entry;  
  }  
  
  public static void UpdateEntry(EntryInfo entry, int userId)  
  {  
    entry.LastModifiedByUserID = userId;  
    entry.LastModifiedDate = DateTime.Now;  
    using (IDataContext ctx = DataContext.Instance())  
    {  
      var rep = ctx.GetRepository<EntryInfo>();  
      rep.Update(entry);  
    }  
  }  
  
  public static void DeleteEntry(EntryInfo entry)  
  {  
    using (IDataContext ctx = DataContext.Instance())  
    {  
      var rep = ctx.GetRepository<EntryInfo>();  
      rep.Delete(entry);  
    }  
  }  
}  
}
```

As you can see, there is not much to it to code the CRUD operations. We've added a bit of logic to do the handling of the audit fields in these methods. Otherwise, it'd be even terser.

We now add an object to hold the view. It is the same as the `EntryInfo` in the previous listing but with the display names of the creating and modifying users. We'll call it `EntryViewInfo.cs`, as shown in [Listing 13.7](#).

[Listing 13.7](#): EntryViewInfo.cs

```
using DotNetNuke.ComponentModel.DataAnnotations;
```



```

namespace WROX.Modules.Guestbook.Components
{
    [Scope("ModuleId")]
    [TableName("vw_WROX_Guestbook_Entries")]
    public class EntryViewInfo : EntryInfo
    {
        public string CreatedByUserDisplayName { get; set; }
        public string LastModifiedByUserDisplayName { get; set; }
    }
}

```

We can now add the method to retrieve these from the database in the EntryController:

```

public static IEnumerable<EntryViewInfo> GetEntries(
    int moduleId,
    bool includeNonApproved)
{
    IEnumerable<EntryViewInfo> entries;

    using (IDataContext ctx = DataContext.Instance())
    {
        var rep = ctx.GetRepository<EntryViewInfo>();
        entries = rep.Get(moduleId);
    }
    if (!includeNonApproved)
    {
        entries = entries.Where(e => e.Approved);
    }

    return entries.OrderByDescending(e => e.CreatedOnDate);
}

```

Finally, we need a method to flag an entry as approved without setting the LastModified user to whoever approves the message. We also add this to the controller:

```

public static void Approve(EntryInfo entry)
{
    entry.Approved = true;
    using (IDataContext ctx = DataContext.Instance())
    {
        var rep = ctx.GetRepository<EntryInfo>();
        rep.Update(entry);
    }
}

```

This concludes the code we need to store and retrieve our data. We're now

going to code the settings for the module.

Settings

Most applications need to allow administrators to manage some parameters. In our example, we included settings because this is so common and so important to do right. We have just one setting regardless of whether we should have an approval mechanism. We call the setting `AutoApprove` internally. It's a Boolean that, when true, causes any added message (you guessed it) to be automatically approved. We need to solve the following parts:

- Persist this setting somewhere.
- Show the administrator a screen with a checkbox and some explanation.
- Make sure the setting is available throughout the UI code.

To start there is a table in DNN that is meant just for this purpose: `ModuleSettings`. It is a table with name/value pairs bound to a module's ID. So our module needs to serialize any setting to the (string) value field and deserialize it when reading back the value. To avoid this bleeding into various classes of your module, we prefer to abstract this into a single settings class. For our module this class will look like [Listing 13.8](#).

[Listing 13.8](#): GuestbookSettings.cs

```
using System.Collections;
using DotNetNuke.Collections;
using DotNetNuke.Entities.Modules;
using DotNetNuke.Common.Utilities;

namespace WROX.Modules.Guestbook.Components
{
    public class GuestbookSettings
    {
        private int ModuleId { get; set; }
        private Hashtable AllSettings { get; set; }

        public bool AutoApprove { get; set; }

        public GuestbookSettings(int moduleId)
        {
            ModuleId = moduleId;
            AllSettings = (new
```

```

ModuleController()).GetModuleSettings(moduleId);
    AutoApprove = AllSettings.GetValueOrDefault("AutoApprove",
false);
}

public static GuestbookSettings GetGuestbookSettings(int
moduleId)
{
    var cacheKey = "WROX.Modules.Guestbook.Settings" + moduleId;
    var settings = DataCache.GetCachedData<GuestbookSettings>(
        new CacheItemArgs(cacheKey),
        args => new GuestbookSettings(moduleId));
    return settings;
}

public void SaveSettings()
{
    var objModules = new ModuleController();
    objModules.UpdateModuleSetting(ModuleId, "AutoApprove",
AutoApprove.ToString());
    var cacheKey = "WROX.Modules.Guestbook.Settings" + ModuleId;
    DataCache.SetCache(cacheKey, this);
}
}
}

```

The constructor reads all the module's settings into a `Hashtable`. It then parses out our `AutoApprove` setting from this using an extension method included in DNN. If the value is not present or it can't be converted to a `Boolean`, it is set to `false`. This takes care of the new module scenarios where a user has just added a module and there are no settings yet. If you try to compile the project at this point, an error about needing to reference `System.Xml.Linq` appears, because of the extension method that was used. Go ahead and add it (from the Assemblies tab in the Add Reference dialog), and that issue should go away.

The static method to create the settings object does an important thing besides calling the constructor: it handles caching. This means that every time the settings are called, they are not being retrieved and reparsed over and over again. Finally, the `SaveSettings` method writes the settings to DNN's `ModuleSettings` table and resets the cache.

It may seem a bit of an overkill for just a single `Boolean`, but obviously in more complex modules you're going to have many values that need to be

stored, and using this approach keeps all your logic for this in one place.

Now you need to make this available to your UI. You may have noticed two classes in the root folder of your module called `GuestbookModuleBase.cs` and `GuestbookModuleSettingsBase.cs`. These were created by your template. If you use another method to create the module, you may need to add these yourself. We consider it good practice to use a class that underlies all your UI controls where you can handle settings and other shared properties.

Commonly, two inheritance chains are found in a module: `PortalModuleBase` and `ModuleSettingsBase`. The former is used for any control that loads to represent your module on the DNN page. The latter is used for a control you use for settings management. You can use this to hook into the module settings screen where it is shown as a tab. This makes for a very consistent user experience. What the `GuestbookModuleBase` and `GuestbookModuleSettingsBase` do is to inherit from these so our module's controls can inherit from them. You can clear out any code that was previously there (change the namespace from `Christoc` to `WROX`) and add to each of them:

```
public new GuestbookSettings Settings
{
    get
    {
        return GuestbookSettings.GetGuestbookSettings(ModuleId);
    }
}
```

The `new` keyword is needed here because you're shadowing `Settings` from the base class, which is the `Hashtable` straight from `ModuleSettings` that you've now encapsulated into your own settings class. You can now access your setting by using `Settings.AutoApprove` at any place in your control code.

So begin by making the settings control that will be loaded in your module. In the manifest of your module that you loaded into DNN, you specified that one control (`Settings.ascx`) had a control key of `Settings`. This is a special case that tells DNN it should attempt to load this control into the module settings screen under its own tab.

You can open `Settings.ascx` and replace whatever is there with the code in [Listing 13.9](#).

[Listing 13.9](#): Settings.ascx

```

<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="Settings.ascx.cs"
    Inherits="WROX.Modules.Guestbook.Settings" %>
<%@ Register TagName="label" TagPrefix="dnn"
Src="~/controls/labelcontrol.ascx" %>

<fieldset>
    <div class="dnnFormItem">
        <dnn:Label ID="lblAutoApprove" runat="server"
ResourceKey="lblAutoApprove"
            ControlName="chkAutoApprove"/>
        <asp:CheckBox runat="server" ID="chkAutoApprove"/>
    </div>
</fieldset>

```

As you can see, a `dnn:Label` control is used here, which is placed before the check box and displays not just a (meaningful) caption but also a help icon that pops up a help text when clicked. This has all been taken care of for you. All you need to do is add the value in the `ResourceKey` attribute to the `Settings.ascx.resx` file ([Figure 13.16](#)) in the correct way. In the example, you need to add `lblAutoApprove.Text` and `lblAutoApprove.Help` as shown in [Figure 13.16](#).

	Name	Value
▶	ControlTitle_settings.Text	Guestbook Settings
	lblAutoApprove.Help	Selecting the checkbox will allow users to add a message without anyone having to approve it
	lblAutoApprove.Text	Auto Approve
*		

[Figure 13.16](#)

You'll also notice the `ControlTitle_settings.Text`. This is used for our title of the tab in the module settings screen. Now open up the code behind for `settings.ascx`. There are two methods to override in this control: `LoadSettings` and `SaveSettings`. These are called by DNN when it uses this control. Given that you're inheriting from the `GuestbookModuleSettingsBase`, your code is fairly simple (see [Listing 13.10](#)).

[Listing 13.10: Settings.ascx.cs](#)

```

using System;
using DotNetNuke.Services.Exceptions;

```

```

namespace WROX.Modules.Guestbook
{
    public partial class Settings : GuestbookModuleSettingsBase
    {
        #region Base Method Implementations

        public override void LoadSettings()
        {
            try
            {
                if (Page.IsPostBack == false)
                {
                    chkAutoApprove.Checked = Settings.AutoApprove;
                }
            }
            catch (Exception exc) //Module failed to load
            {
                Exceptions.ProcessModuleLoadException(this, exc);
            }
        }

        public override void UpdateSettings()
        {
            try
            {
                Settings.AutoApprove = chkAutoApprove.Checked;
                Settings.SaveSettings();
            }
            catch (Exception exc) //Module failed to load
            {
                Exceptions.ProcessModuleLoadException(this, exc);
            }
        }

        #endregion
    }
}

```

This completes the work to implement your settings. Administrators can now manage settings in the module's settings panel, and this is persisted to all controls throughout the module. We now move to the UI for your module.

The UI

Regular users will see only two controls you create: `View.ascx` and `Edit.ascx`. `View.ascx` is the default control that is loaded whenever the user comes to the page (it has no control key in the definition). `Edit` is where you allow the user to either create or edit a message. It has the control key "Edit".

There are two properties to which you need to regularly refer: the currently selected `EntryId` and whether the user is allowed to edit other people's messages. Abstracting these two properties into your base class allows you to isolate that logic and save on the amount of code that is added in both the `view` and `edit` controls.

Examine the query string to determine the `EntryId`. You can do this as follows in `GuestbookModuleBase` (you will also need to add `using DotNetNuke.Collections` for the `GetValueOrDefault` extension method):

```
protected override void OnInit(EventArgs e)
{
    _entryId = Request.Params.GetValueOrDefault("EntryId", -1);
}

private int _entryId;
public int EntryId
{
    get
    {
        return _entryId;
    }
}
}
```

Whether the user has edit permissions is determined by the `ModulePermissionController` (from the `DotNetNuke.Security.Permissions` namespace) depending on the module's permissions collection:

```
public bool CanEdit
{
    get
    {
        return ModulePermissionController.HasModulePermission(
            ModuleConfiguration.ModulePermissions, "EDIT");
    }
}
```

These two properties are now at your fingertips when writing out the `View` and `Edit` controls. The `View` control has a repeater to show the messages and an `Add` button that users can click to create a new message. It will look like [Listing 13.11](#).

[Listing 13.11: View.ascx](#)

```

<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="View.ascx.cs"
    Inherits="WROX.Modules.Guestbook.View" %>
<%@ Import Namespace="WROX.Modules.Guestbook.Components" %>

<div>
    <asp:Repeater runat="server" ID="rpGuestbook"
        OnItemDataBound="rpGuestbook_ItemDataBound"
        OnItemCommand="rpGuestbook_ItemCommand">
        <ItemTemplate>
            <div class="row-fluid messageRow">
                <div class="span3">
                    <h3>
                        <%# Eval("CreatedByUserDisplayName") %>
                    </h3>
                    <p><%# ((DateTime)Eval("CreatedOnDate")).ToString("D") %></p>
                </div>
                <div class="span9 message">
                    <div>
                        <%# Eval("Message") %>
                    </div>
                    <div class="messageButtons">
                        <asp:HyperLink ID="cmdEdit" runat="server"
ResourceKey="cmdEdit"
                            Visible="false" Enabled="false" CssClass="btnMessage"/>
                        <asp:LinkButton ID="cmdApprove" runat="server"
ResourceKey="cmdApprove"
                            Visible="false" Enabled="false" CommandName="Approve"
                            CssClass="btnMessage"/>
                        <asp:LinkButton ID="cmdDelete" runat="server"
ResourceKey="cmdDelete"
                            Visible="false" Enabled="false" CommandName="Delete"
                            CssClass="btnMessage"/>
                    </div>
                </div>
                <div class="editNote" style="display:<%#
((DateTime)Eval("CreatedOnDate"))
                    == (DateTime)Eval("LastModifiedOnDate") ? "none" :
"block"> %>>
                    <%# EditString((EntryViewInfo)Container.DataItem) %>
                </div>
            </div>
        </ItemTemplate>
    </asp:Repeater>
</div>

<asp:HyperLink runat="server" ID="cmdAdd"
CssClass="dnnPrimaryAction">
    <%=LocalizeString("cmdAdd") %>
</asp:HyperLink>

```


As the default DNN 7 skin is used for this example, you can use the Bootstrap 2 classes that are loaded. In a real-world scenario you might need to either design it for a specific skin or create all of your CSS to include in the module. This is not trivial, and practice will tell you how best to approach this.

You'll notice that three buttons have been defined for each message row. The visibility and events for these buttons are handled by the code behind of this control (see [Listing 13.12](#)).

[Listing 13.12](#): View.ascx.cs

```
using System;
using System.Web.UI.WebControls;
using WROX.Modules.Guestbook.Components;
using DotNetNuke.Services.Exceptions;
using DotNetNuke.Services.Localization;
using DotNetNuke.UI.Utilities;

namespace WROX.Modules.Guestbook
{
    public partial class View : GuestbookModuleBase
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            try
            {
                cmdAdd.NavigateUrl = EditUrl("Edit");
                var showAll = Settings.AutoApprove;
                if (!showAll)
                {
                    showAll = CanEdit;
                }
                rpGuestbook.DataSource = EntryController.GetEntries(ModuleId,
showAll);
                rpGuestbook.DataBind();
            }
            catch (Exception exc) //Module failed to load
            {
                Exceptions.ProcessModuleLoadException(this, exc);
            }
        }

        protected void rpGuestbook_ItemDataBound(object sender,
RepeaterItemEventArgs e)
        {
            if (e.Item.ItemType == ListItemType.AlternatingItem
                || e.Item.ItemType == ListItemType.Item)
            {
```

```

var cmdEdit = e.Item.FindControl("cmdEdit") as HyperLink;
var cmdDelete = e.Item.FindControl("cmdDelete") as LinkButton;
var cmdApprove = e.Item.FindControl("cmdApprove") as
LinkButton;

var entry = (EntryViewInfo)e.Item.DataItem;

if (cmdDelete != null && cmdApprove != null && cmdEdit !=
null)
{
cmdDelete.CommandArgument = entry.EntryId.ToString();
cmdApprove.CommandArgument = entry.EntryId.ToString();
cmdEdit.NavigateUrl = EditUrl("EntryId",
entry.EntryId.ToString(), "Edit");
ClientAPI.AddButtonConfirm(cmdDelete,
LocalizeString("ConfirmDelete"));
cmdApprove.Enabled = cmdApprove.Visible =
!Settings.AutoApprove
&& !entry.Approved && CanEdit;
if (Settings.AutoApprove)
{
cmdDelete.Enabled = cmdDelete.Visible = cmdEdit.Enabled =
cmdEdit.Visible
= (CanEdit || (entry.CreatedByUserID == UserId && UserId
!= -1));
}
else
{
cmdDelete.Enabled = cmdDelete.Visible = cmdEdit.Enabled =
cmdEdit.Visible
= CanEdit;
}
}
}

protected void rpGuestbook_ItemCommand(object source,
RepeaterCommandEventArgs e)
{
if (e.CommandName == "Delete")
{
var entry =
EntryController.GetEntry(Convert.ToInt32(e.CommandArgument),
ModuleId);
if (entry != null)
{
EntryController.DeleteEntry(entry);
}
}
if (e.CommandName == "Approve")
{

```

```

    var entry =
EntryController.GetEntry(Convert.ToInt32(e.CommandArgument),
                        ModuleId);
    if (entry != null)
    {
        EntryController.Approve(entry);
    }
}
Response.Redirect(DotNetNuke.Common.Globals.NavigateURL());
}

public string EditString(EntryViewInfo entry)
{
    return string.Format(LocalizeString("Edited"),
        entry.LastModifiedByUserDisplayName,
entry.LastModifiedOnDate);
}
}
}

```

Note the use of `EditUrl` in the logic to create the edit link and in the handler of the `add` command. `EditUrl` is a DNN function that constructs a URL that loads the edit screen. Optionally, it can receive parameters that allow the edit screen to determine if it is editing an existing entry or creating a new one. In this example, `EntryId=X` is used for this, where X is the entry ID of the entry. If it's omitted, assume it is a new entry you should be creating.

Now let's create the edit screen. The edit screen should have a text box (an HTML editor can be used, but for this example we restrict it to text only), a submit button, and a cancel button (see [Listing 13.13](#)).

[Listing 13.13: Edit.ascx](#)

```

<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="Edit.ascx.cs"
    Inherits="WROX.Modules.Guestbook.Edit" %>
<h3>
    <asp:Label runat="server" ID="lblMessage"
ResourceKey="lblMessage"/>
</h3>
<div>
    <asp:TextBox ID="txtMessage" runat="server" TextMode="MultiLine"
        Rows="5" Width="50%"/>
</div>
<div runat="server" id="divApproveWarning" class="dnnFormMessage

```

```

dnnFormWarning">
  <asp:label runat="server" ID="lblApproveWarning"
    ResourceKey="lblApproveWarning"/>
</div>
<p>
  <asp:LinkButton ID="btnSubmit" runat="server"
OnClick="btnSubmit_Click"
  ResourceKey="btnSubmit" CssClass="dnnPrimaryAction"/>
  <asp:LinkButton ID="btnCancel" runat="server"
OnClick="btnCancel_Click"
  ResourceKey="btnCancel" CssClass="dnnSecondaryAction"/>
</p>

```

Note that a message that can be switched on or off is added to tell the user that a new message needs to be approved to be visible (see [Listing 13.14](#)).

[Listing 13.14: Edit.ascx.cs](#)

```

using System;
using DotNetNuke.Entities.Users;
using DotNetNuke.Security;
using WROX.Modules.Guestbook.Components;
using DotNetNuke.Services.Exceptions;

namespace WROX.Modules.Guestbook
{
  public partial class Edit : GuestbookModuleBase
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      try
      {
        if (!Page.IsPostBack)
        {
          if (EntryId > 0)
          {
            if (UserId == -1)
            {
              throw new Exception("Anonymous users cannot edit
messages");
            }
            var entry = EntryController.GetEntry(EntryId, ModuleId);
            if (!CanEdit)
            {
              if (entry.CreatedByUserID != UserId)
              {
                throw new Exception("You cannot edit someone else's
message");
              }
            }
          }
        }
      }
    }
  }
}

```

```

    }
    }
    txtMessage.Text = entry.Message;
    }
    divApproveWarning.Visible = !Settings.AutoApprove;
    }
}
catch (Exception exc) //Module failed to load
{
    Exceptions.ProcessModuleLoadException(this, exc);
}
}

protected void btnSubmit_Click(object sender, EventArgs e)
{
    var entry = new EntryInfo();
    if (EntryId > 0)
    {
        if (UserId == -1)
        {
            throw new Exception("Anonymous users cannot edit messages");
        }
        entry = EntryController.GetEntry(EntryId, ModuleId);
        if (!CanEdit)
        {
            if (entry.CreatedByUserID != UserId)
            {
                throw new Exception("You cannot edit someone else's
message");
            }
        }
    }
    else
    {
        entry.Approved = Settings.AutoApprove;
        entry.ModuleId = ModuleId;
    }

    entry.Message = (new
PortalSecurity()).InputFilter(txtMessage.Text,
    PortalSecurity.FilterFlag.NoMarkup |
PortalSecurity.FilterFlag.NoSQL);

    if (EntryId > 0)
    {
        EntryController.UpdateEntry(entry, UserId);
    }
    else
    {
        EntryController.AddEntry(entry, UserId);
    }
}

```

```

    }
    Response.Redirect(DotNetNuke.Common.Globals.NavigateURL());
}

protected void btnCancel_Click(object sender, EventArgs e)
{
    Response.Redirect(DotNetNuke.Common.Globals.NavigateURL());
}
}
}
}

```

Note that quite a bit of code is devoted to security. First, in both the page load and in the handler for the submit button, a number of checks are done to see if the user is allowed to edit an existing message. Even though in regular use the user could never get here (the edit button wouldn't be shown in the view control), the application is steered through the query string, and it would be trivial for someone with knowledge of DNN to construct an edit link that would load the message. Always assume that hackers will know how to create a URL that you depend on and that they are using tools like Fiddler to create requests that resemble what you'd expect from a regular user. The one thing you can rely on is that DNN did its work authenticating the user. This includes matching the user's permissions in the module settings to the control's type. In this case (in the module's manifest you'll see that both the view and edit control are of `controlType View`), the user must have view permissions if he is allowed to see this control.

Second, you'll notice we are passing whatever is added through the message box through `PortalSecurity.InputFilter`. This method allows us to strip various bits from the input that may be leveraged by hackers or spammers to wreak havoc. So we're not allowing any SQL (SQL injection attacks) or JavaScript or even markup. The latter is also to prevent users accidentally messing up the look of the page. But markup could be used by spammers to show links to other sites.

The views have used a number of localized strings, so you'll need to make sure to add those to ensure that the labels show correctly. [Tables 13.3](#) and [13.4](#) show the contents of the resource files that belong to the views.

Table 13.3 View.ascx.resx Entries

Name	Value
cmdEdit.Text	Edit

cmdApprove.Text	Approve
cmdDelete.Text	Delete
cmdAdd.Text	Add
Edited.Text	Edited by {0} on {1:d}
ConfirmDelete.Text	Are you sure you want to delete this message?

Table 13.4 Edit.ascx.resx entries

Name	Value
btnCancel.Text	Cancel
btnSubmit.Text	Submit
lblApproveWarning.Text	Your message will not display until it is approved by a moderator
lblMessage.Text	Message

Finally, we add some CSS to `module.css` to make the result look acceptable. The `module.css` file is loaded automatically by DNN, after the framework's default CSS but before the CSS that comes with the skin. That way, a designer can override specific modules' CSS classes for a particular skin. You learn more about DNN's cascading model in [Chapter 17](#), "Skinning." [Listing 13.15](#) shows the CSS that is used for the sample module.

[Listing 13.15: Module.css](#)

```
.messageRow {
border: 1px solid #ddd;
border-width: 0 0 1px 0;
margin-bottom: 20px;
}
.editNote {
font-size: 75%;
color: #999;
width: 100%;
text-align: right;
line-height: 1.5em;
}
.message {
padding-bottom: 10px;
}
.btnMessage {
padding: 2px 6px 2px 6px;
margin-right: 6px;
border-radius: 4px;
border: 1px solid #ddd;
background-color: #fff;
```

```
}  
.btnMessage:hover {  
  background-color: #ddd;  
}  
.messageButtons {  
  padding-top: 10px;  
}
```

Results

The module is now ready to run. Your solution should look something like [Figure 13.17](#). You can compile what you have and make sure that the DLL is written to the bin folder of your dev site (you will probably need to go into the project properties and adjust the Output path in the Build tab to `..\..\..\bin\`, to account for the WROX folder). You may notice a delay in the loading of the site as ASP.NET reloads all the DLLs of the site and recycles the app pool. Every time you build to the bin folder, the site completely refreshes, empties all caches, and starts loading from scratch.

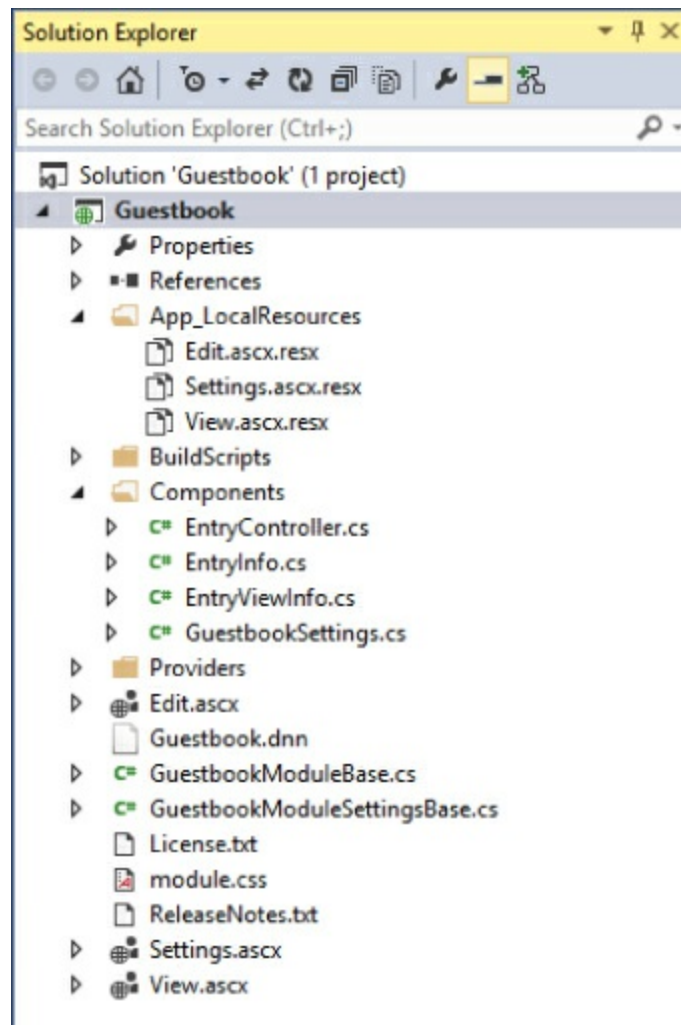
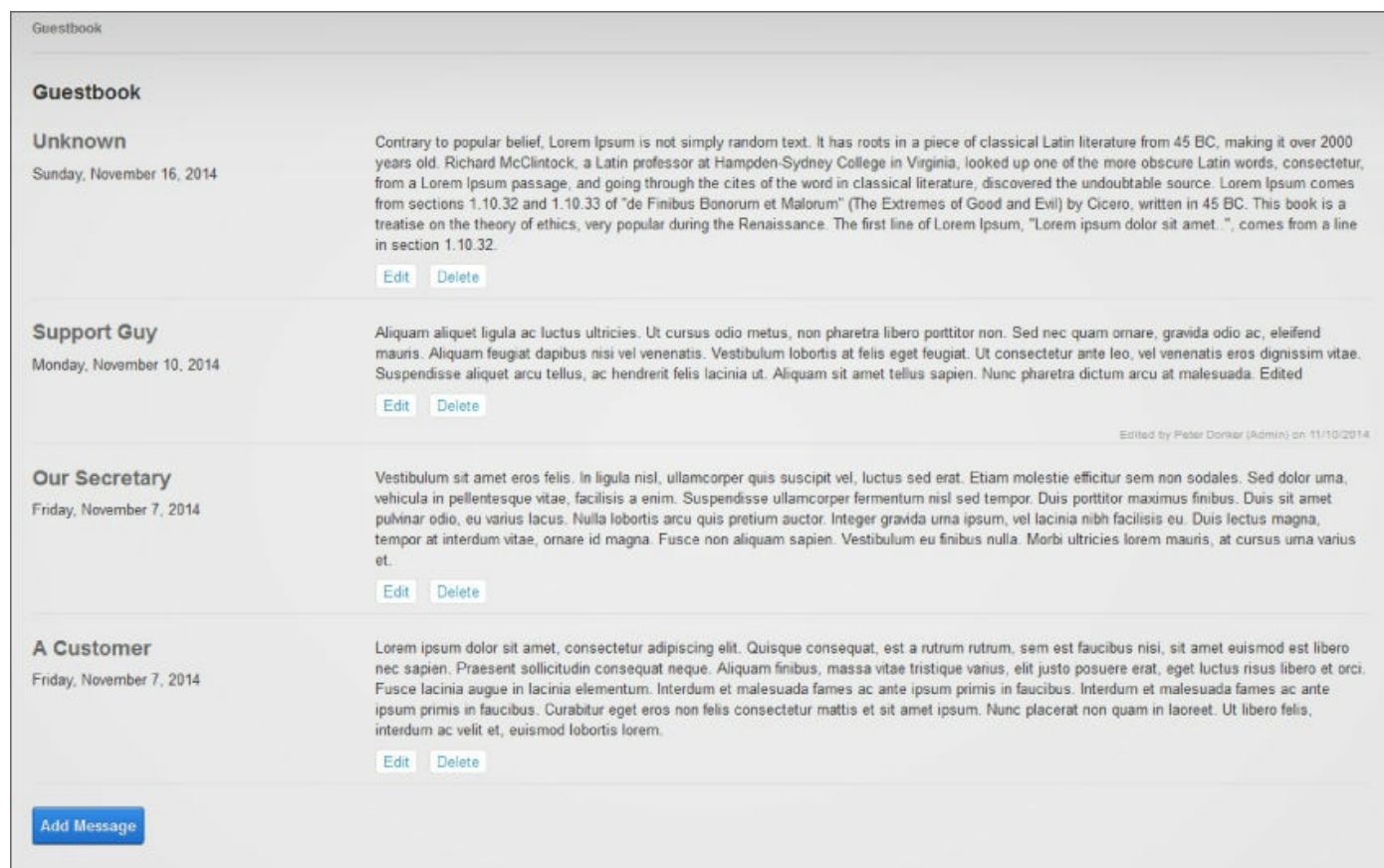


Figure 13.17

You should see your module page with an add button. Add some messages to verify its operation ([Figure 13.18](#)).



The screenshot shows a Moodle Guestbook interface. At the top, there is a header 'Guestbook'. Below it, the title 'Guestbook' is displayed. The main content area contains four messages, each with a title, a date, a body of text, and 'Edit' and 'Delete' buttons. The messages are:

- Unknown**: Sunday, November 16, 2014. Text: 'Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, "Lorem ipsum dolor sit amet.", comes from a line in section 1.10.32.' Buttons: Edit, Delete.
- Support Guy**: Monday, November 10, 2014. Text: 'Aliquam aliquet ligula ac luctus ultricies. Ut cursus odio metus, non pharetra libero porttitor non. Sed nec quam ornare, gravida odio ac, eleifend mauris. Aliquam feugiat dapibus nisi vel venenatis. Vestibulum lobortis at felis eget feugiat. Ut consectetur ante leo, vel venenatis eros dignissim vitae. Suspendisse aliquet arcu tellus, ac hendrerit felis lacinia ut. Aliquam sit amet tellus sapien. Nunc pharetra dictum arcu at malesuada. Edited' Buttons: Edit, Delete. A small note at the bottom right of the message says 'Edited by Peter Donker (Admin) on 11/10/2014'.
- Our Secretary**: Friday, November 7, 2014. Text: 'Vestibulum sit amet eros felis. In ligula nisl, ullamcorper quis suscipit vel, luctus sed erat. Etiam molestie efficitur sem non sodales. Sed dolor uma, vehicula in pellentesque vitae, facilisis a enim. Suspendisse ullamcorper fermentum nisl sed tempor. Duis porttitor maximus finibus. Duis sit amet pulvinar odio, eu varius lacus. Nulla lobortis arcu quis pretium auctor. Integer gravida uma ipsum, vel lacinia nibh facilisis eu. Duis lectus magna, tempor at interdum vitae, ornare id magna. Fusce non aliquam sapien. Vestibulum eu finibus nulla. Morbi ultricies lorem mauris, at cursus uma varius et.' Buttons: Edit, Delete.
- A Customer**: Friday, November 7, 2014. Text: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque consequat, est a rutrum rutrum, sem est faucibus nisi, sit amet euismod est libero nec sapien. Praesent sollicitudin consequat neque. Aliquam finibus, massa vitae tristique varius, elit justo posuere erat, eget luctus risus libero et orci. Fusce lacinia augue in lacinia elementum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Interdum et malesuada fames ac ante ipsum primis in faucibus. Curabitur eget eros non felis consectetur mattis et sit amet ipsum. Nunc placerat non quam in laoreet. Ut libero felis, interdum ac velit et, euismod lobortis lorem.' Buttons: Edit, Delete.

At the bottom left of the page, there is a blue button labeled 'Add Message'.

Figure 13.18

Now switch to edit mode and use the module's menu to pop up the settings. A fourth tab called Guestbook Settings with a check box appears ([Figure 13.19](#)).

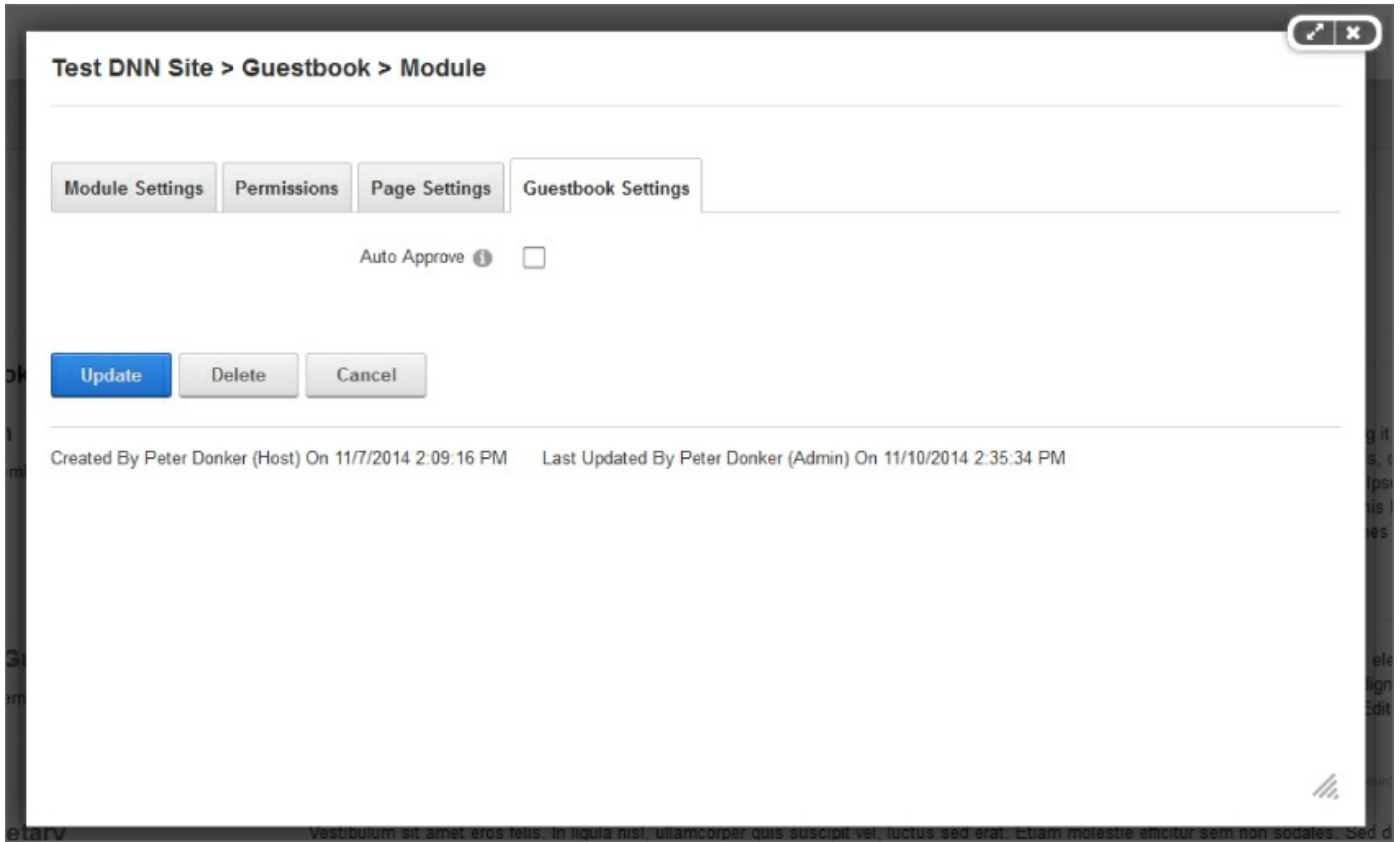


Figure 13.19

Wrapping It Up

Once you're happy with how the module is working, it's time to consider packaging. We've mentioned a few times that modules come as Zip files. But there is a bit more to this.

The Manifest

By far the most crucial part of the module's package is the manifest. The manifest is an XML file with a .dnn extension that tells the DNN installer how to install this module. It tells DNN the following:

- What it's about to install. Is this a module, a skin, or some other type of extension?
- Who is the maker? Name, URL, and so on. You can also include a license blurb here.
- In case it's a module, the definition of the module.
- SQL scripts to run.
- Where to put the files that are in the Zip file.

There are many options for the manifest but you don't need to go through them exhaustively at this stage. Let's look at the manifest for the Guestbook module to get an idea (see [Listing 13.16](#)).

Listing 13.16: Guestbook module manifest

```
<dotnetnuke type="Package" version="5.0">
  <packages>
    <package name="WROX_Guestbook" type="Module" version="01.00.00">
      <friendlyName>Guestbook</friendlyName>
      <description>WROX.com Guestbook module</description>
      <owner>
        <name>WROX.com</name>
        <organization>WROX.com</organization>
        <url>http://www.wrox.com</url>
        <email>modules@wrox.com</email>
      </owner>
      <license src="License.txt"/>
      <releaseNotes src="ReleaseNotes.txt"/>
      <dependencies>
        <dependency type="CoreVersion">07.01.02</dependency>
      </dependencies>
    </package>
  </packages>
</dotnetnuke>
```

```

<components>
  <component type="Script">
    <scripts>
      <basePath>DesktopModules\WROX\Guestbook</basePath>
      <script type="Install">
        <path>Providers\DataProviders\SqlDataProvider</path>
        <name>01.00.00.SqlDataProvider</name>
        <version>01.00.00</version>
      </script>
      <script type="UnInstall">
        <path>Providers\DataProviders\SqlDataProvider</path>
        <name>Uninstall.SqlDataProvider</name>
        <version>01.00.00</version>
      </script>
    </scripts>
  </component>
  <component type="ResourceFile">
    <resourceFiles>
      <basePath>DesktopModules\WROX\Guestbook</basePath>
      <resourceFile>
        <name>Resources.zip</name>
      </resourceFile>
    </resourceFiles>
  </component>
  <component type="Module">
    <desktopModule>
      <moduleName>WROX_Guestbook</moduleName>
      <foldername>WROX\Guestbook</foldername>
      <moduleDefinitions>
        <moduleDefinition>
          <friendlyName>Guestbook</friendlyName>
          <defaultCacheTime>0</defaultCacheTime>
          <moduleControls>
            <moduleControl>
              <controlKey/>

<controlSrc>DesktopModules/WROX/Guestbook/View.ascx</controlSrc>

<supportsPartialRendering>False</supportsPartialRendering>
      <controlTitle/>
      <controlType>View</controlType>
      <viewOrder>0</viewOrder>
    </moduleControl>
    <moduleControl>
      <controlKey>Edit</controlKey>

<controlSrc>DesktopModules/WROX/Guestbook/Edit.ascx</controlSrc>

<supportsPartialRendering>False</supportsPartialRendering>
      <controlTitle>Edit Content</controlTitle>
      <controlType>View</controlType>

```

```

        <viewOrder>0</viewOrder>
        <supportsPopUps>False</supportsPopUps>
    </moduleControl>
    <moduleControl>
        <controlKey>Settings</controlKey>
<controlSrc>DesktopModules/WROX/Guestbook/Settings.ascx</controlSrc>

<supportsPartialRendering>False</supportsPartialRendering>
    <controlTitle>Guestbook Settings</controlTitle>
    <controlType>Edit</controlType>
    <viewOrder>0</viewOrder>
    </moduleControl>
</moduleControls>
</moduleDefinition>
</moduleDefinitions>
</desktopModule>
</component>
<component type="Assembly">
    <assemblies>
        <assembly>
            <name>WROX.Modules.Guestbook.dll</name>
            <path>bin</path>
        </assembly>
    </assemblies>
</component>
</components>
</package>
</packages>
</dotnetnuke>

```

The Preamble

The `package` node begins by telling DNN it's a module and which version this is. DNN uses this and the name attribute (`WROX_Guestbook`) to determine if the package has already been installed. So you should never ever change this name attribute or you will break your upgrade sequence!

The opening tag is followed by a number of tags that have a vanity role. The friendly name is what is shown in the UI as a name for the module, and the description is shown on the Extensions page. The owner information and license/release notes are shown during installation.

The dependency node you see makes sure that users who try to install this on a version older than DNN 7.1.2 will not be able to do so. DNN informs them and stops the installation procedure. As pointed out at the start of the

chapter, if the installation proceeded, the result would be a big mess within .NET as a dependency of our module's DLL on a higher version of the DotNetNuke.dll than the one installed will break our module entirely.

So how about other dependencies? In our module we've also used a dependency on `DotNetNuke.WebUtility`. But because this ships with DNN, it can be safely assumed that it will work out fine. In other words, you only need to worry about dependencies on components that do not ship with DNN.

SQL Scripts

The node `<component type="Script">` enumerates all SQL scripts. These are incremental, and DNN uses the module name and the version number to figure out which scripts it should run. It should be clear that you should have a solid version numbering strategy if you are going to distribute the module. Errors in scripts are not recoverable! If DNN somehow trips on one of those scripts, the whole installation fails and you are stuck with a situation where some (parts of) scripts may have run and others have not. This is potentially catastrophic, so the only safe way to install modules is to back up your DNN installation first. Make sure to mention this to your audience when releasing an upgrade. It is up to you to make these upgrade scripts as solid as you can.

The scripts are named `xx.yy.zz.SqlDataProvider` by convention. You can opt to make new installs start somewhere down the script history by adding an `Install.xx.yy.zz.SqlDataProvider` script. This is not uncommon for long-lived modules. If you examine the manifest of the Blog module 6.0.x, you'll notice this is done at version 6. So there is an `Install.06.00.00.SqlDataProvider` script that makes new installs start there and continue up. If it's an upgrade, all `Install.*` scripts are ignored.

You should also include the `UnInstall` script. There is no version number for this (DNN does not support incremental uninstall scripts), so we don't include this in the name. Curiously, it's expected in the manifest "version" element, but the value is not used by any of DNN's logic. The convention is to use the current module's version for this value. During each upgrade the latest version is written over what was previously there. DNN then uses this script during the removal of the module if the user decides to uninstall. Again, if you're distributing to a wider audience, spend some time perfecting these scripts. Make sure you make these scripts robust so there is little chance they fail. For example, test if a procedure is there before you drop it. It may seem redundant, but a failure in the script can leave the uninstall

hanging in midair, and users will bombard you with (angry) emails.

The Resource File

A very common technique to deliver a bunch of files to the target DNN installation is to use a resources file. This is a Zip file containing the whole tree of folders and files you want to distribute. DNN unpacks it to the module folder. It saves you a lot of work announcing each individual file to install. Instead, your approach changes to which files can go into the resource file and which ones must be kept outside. The answer is that all DLLs, SQL scripts, and cleanup files must stay outside, as they are not simply unzipped to the destination. Everything else can be zipped up.

The Module Definition

Earlier in this chapter we looked at how modules are embedded in DNN. It will come as no surprise that there is a reflection of that in the manifest. It is in the `<component type="Module">` node of the manifest. Walking through that you will recognize most of the fields of the relevant tables. This is where you tell DNN what your module is called, where it should go, which features it supports, the definition(s) and their controls, and so on.

There is an element in the definition that hasn't been touched on because it's not included here, but you can find it in the manifest of the standard HTML module: `eventMessage`. This tells DNN to have the module notified of some events. Typically you'll see an entry here to listen to "upgrade," as is the case here. It is quite possible you can't do all your upgrade processing with SQL scripts (like manipulation of files, for example). What this does is that DNN will run the upgrade method on the type supplied under `businessControllerClass` based on the list of versions in `upgradeVersionsList`. You can use this method to embed logic in your module to do upgrades internally that require code to run.

Assemblies

The node `<component type="Assembly">` lists all DLLs to be installed. You actually need to specify the bin folder as there are scenarios where DLLs can go elsewhere. Using the `version` attribute you can make sure DNN keeps track of versions of your DLLs. This can be leveraged with shared DLLs. Let's say you use `Acme.dll` and have compiled your module using version 1.1.0 from this company, then using the version code will prevent another module

developer using the same Acme.dll but with version 1.0.0 to get that DLL installed over yours as that might break your code. Note we assume that shared DLLs always support backward compatibility, which is kind of an industry standard.

Packing Up the Guestbook Module

To wrap it all up, all we need to do is zip up our work:

1. Create a Zip file called resources.zip. In it we put
 - The three ascx files (View, Edit, Settings)
 - module.css
 - App_LocalResources*.resx (preserve the folder in the zip file!)
2. Create a second zip file that incorporates
 - The resources.zip file you just created
 - License.txt and ReleaseNotes.txt
 - The manifest (Guestbook.dnn)
 - Bin\Guestbook.dll
 - Providers\DataProviders\SqlDataProvider*.SqlDataProvider

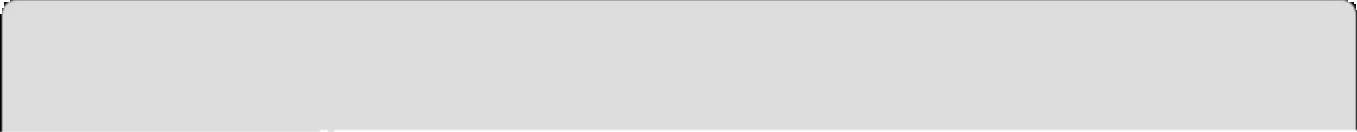
This second zip file is your module. The convention is to give it a name along these lines: [ModuleName]_[Version]_[Install/Source].zip, so in the previous case Guestbook_01.00.00_Install.zip. That immediately makes it clear to anyone familiar with DNN what the Zip file is.

Creating a Source Version

It is fairly easy to create a source code distribution of the module using the same method as what was just shown. The only difference is that you'd include all necessary source code files in the resources.zip. The manifest would not need to be changed because the resources file is simply unpacked. But it would already prime the receiving DNN installation with SQL scripts and the module definition so that anyone wanting to work on the source would just need to fire up Visual Studio, load up your project, and recompile to the site's bin folder when needed. This makes it incredibly easy for someone else to work with your source code.

Build Automation

At this point you may think that all the packing and manifest creation just described is quite a bit of work. The good news is that this is very repetitive and that all other module developers go through the same process. As a result, various mechanisms have surfaced to automate this process. The most common way is to use MSBuild tasks to do this. MSBuild is responsible for what happens when you click Build in Visual Studio. But it's not just about invoking the compiler. It can be extended to do all kinds of things, both before and after compilation. This is modeled after Ant and NAnt (the .NET version of Ant). So like NAnt, it, too, uses XML files to tell it what to do. Because MSBuild comes with .NET, it is very tightly integrated with your whole environment, which many see as a bonus. Personally, we use NAnt because it can be kept totally isolated from the project. But it's a matter of taste which build automation technology you use. Just know that once you have this tuned correctly, making a distributable module becomes trivially easy.



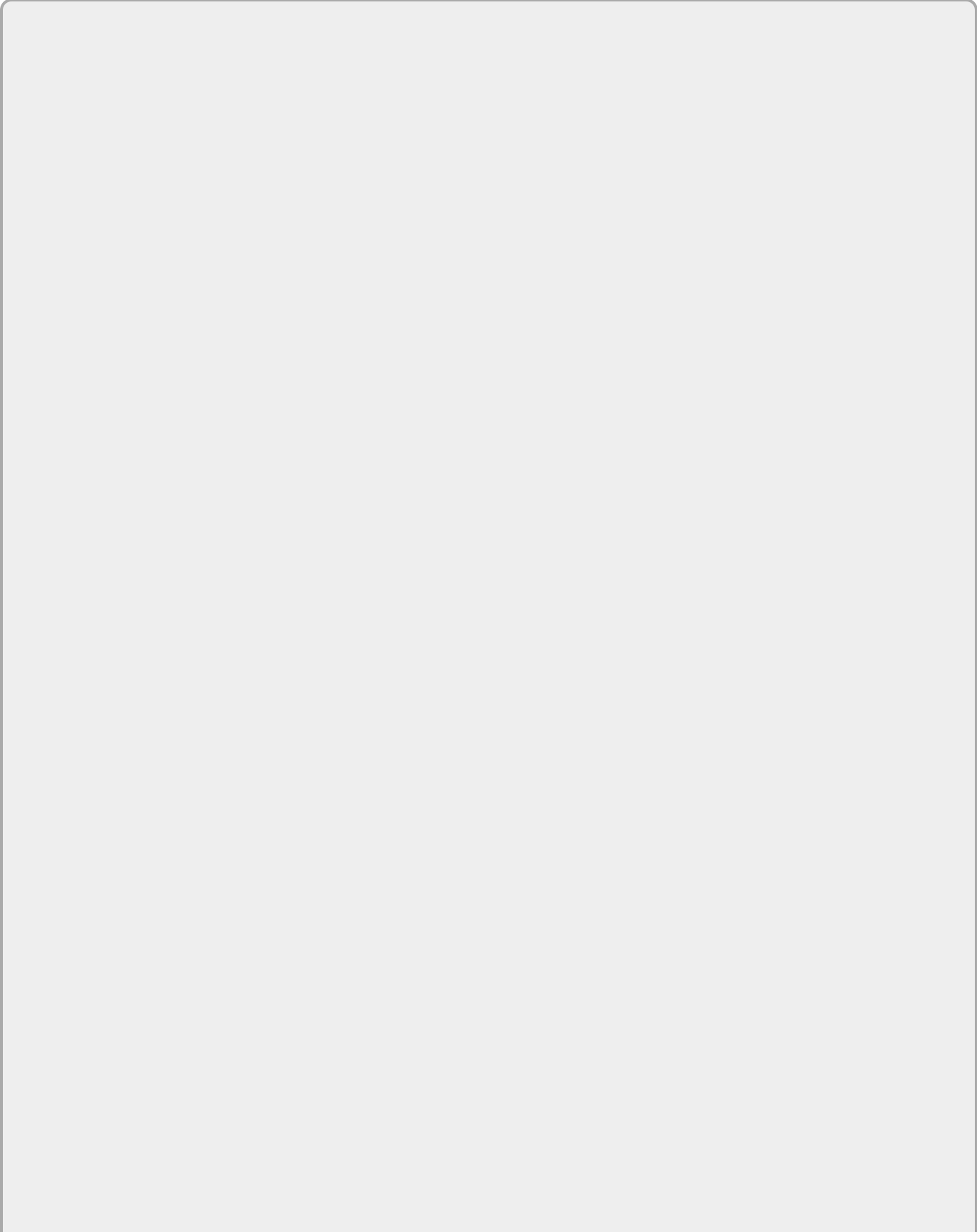
NOTE

Hammond's templates that we used in this chapter include the necessary MSBuild tasks. Building the project in release mode will actually create the module package.

Summary

In this chapter, you've seen a variety of ways to create modules in DNN, both inline and externally developed. You've also taken a close look at how the mechanism works that renders your work inside the DNN platform looking at the relationship between packages, modules, tab modules, and other DNN core objects. Finally, we created a guestbook module and took a closer look at some of the basics of making a DNN module. This knowledge should be largely sufficient to get you started on module development.

Chapter 14
Developing Modules: User Interfaces



What You Will Learn in This Chapter

- Understanding module/page interactions
- Supporting AJAX in modules
- Using jQuery and jQueryUI
- Referencing and managing custom JavaScript
- Using DNN jQuery plugins
- Implementing consistent design

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 14](#) download and individually named according to the names throughout the chapter.

[Chapter 13](#) introduced a plethora of information and is designed to be a high-level introduction to the ins and outs of developing modules using the DNN Platform. This chapter starts a deep dive into the specifics and nuances of working with the DNN environment as it relates to the user interface portion of your module. We will revisit a few topics from prior chapters to add a deeper level of information and context to each of these elements.

Understanding DNN and Module Interactions

Before diving into any of the specific development patterns or even the specific UI controls and other elements, it is important to understand how a DNN module is used. This is important as, unlike development when you have full control over the page, the site administrator has that level of control in DNN.

In the prior chapter, we introduced the concept of Tabs, Modules, and TabModules as they relate to module development. The Tab and its related ID value reference the page of content, the Module represents the module instance, and the TabModule represents the placement of the module on the page. These are important for developers to understand because it is possible that users of your modules will add more than one instance of your module to a page at the same time.

Using [Figure 14.1](#) as an example, you see a DNN site with three different text HTML modules. For illustration's sake, imagine that “Welcome” is `ModuleId 450`, “I Like Words” is `ModuleId 451`, and “More Words Too” is `ModuleId 452`.

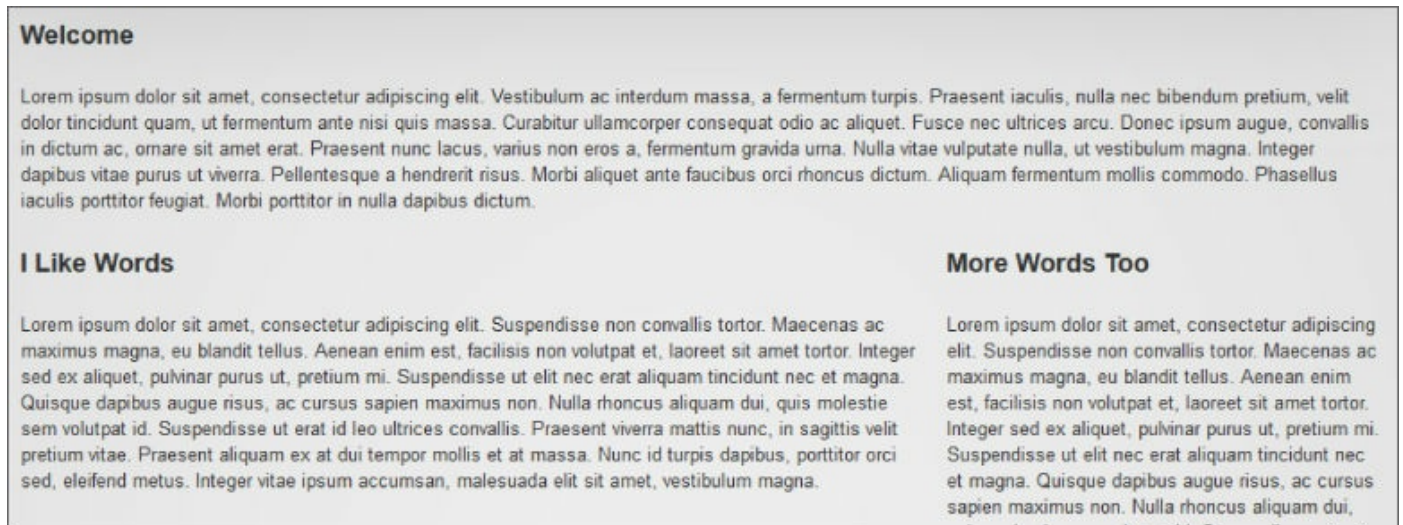


Figure 14.1

As a developer working on the user interface for a module, this is important, as you need to ensure that no conflicts exist in the methods that you create or the JavaScript that you write. When working with module controls and creating functions that interact, be sure to test your module in situations similar to the one explained here to ensure that having the user interact with one instance of your module on a page impacts only that particular module.

A simple solution is to ensure that any JavaScript actions utilize an element

ID that is truly unique to the module. An example might be
`abd_[ModuleId]_[ItemId]`. This approach allows you to easily isolate your
information based on the particular elements in question.

Dialogs and AJAX Support

Modern-day web users are demanding more and more from website user interfaces. Users want to interact with a website in the most efficient manner possible and want to see the information in a meaningful manner. There are two quick options available for module developers to improve the responsiveness and utility of their applications—simple AJAX support and modal dialogs.

DNN AJAX Support

ASP.NET AJAX has been around for a long time, and DNN has built-in support for it. A quick and easy way to make a module more responsive to users is to have it wrapped in an `UpdatePanel`, so that when it needs to trigger a postback action, only the module is updated. This improves user experiences by improving the speed of page load. The users also retain their focus on the section of the page that they are on without a full-screen flicker. [Listing 14.1](#) shows an excerpt of the module manifest from [Chapter 13](#).

[Listing 14.1](#): Guestbook Module Manifest Updates

```
<moduleControl>
  <controlKey/>
  <controlSrc>DesktopModules/WROX/Guestbook/View.ascx</controlSrc>
  <supportsPartialRendering>True</supportsPartialRendering>
  <controlTitle/>
  <controlType>View</controlType>
  <viewOrder>0</viewOrder>
</moduleControl>
<moduleControl>
  <controlKey>Edit</controlKey>
  <controlSrc>DesktopModules/WROX/Guestbook/Edit.ascx</controlSrc>
  <supportsPartialRendering>False</supportsPartialRendering>
  <controlTitle>Edit Content</controlTitle>
  <controlType>View</controlType>
  <viewOrder>0</viewOrder>
  <supportsPopUps>False</supportsPopUps>
</moduleControl>
```

To wrap the `View.ascx` control in an `UpdatePanel`, you need to change the `supportsPartialRendering` XML node from `False` to `True`. This instructs DNN to include this control inside an `UpdatePanel` at runtime. No further

interaction is necessary.

If you find yourself needing to trigger a full postback when using this method of AJAX support, for example using an Export button, you can use the code shown in [Listing 14.2](#).

[Listing 14.2](#): Register Postback Control

```
DotNetNuke.Framework.AJAX.RegisterPostBackControl(myExportButton);
```

This code snippet assumes the control named `myExportButton` will trigger a full postback to the server. For those looking for more granular control of what is included in an `UpdatePanel`, there are methods on the `AJAX` class referenced in [Listing 14.2](#) that allow you to wrap objects in `UpdatePanels`.

DNN Modal Dialogs

Since DNN 6.0.0, modal dialogs have been an integral part of the interface and have become second nature for those navigating DNN-based sites. Modal dialogs are great for displaying user interfaces that appear after a user has selected something in a module and you want other content to stay visible behind the scenes.

[Figure 14.2](#) shows the most recognizable use—the default DNN login page.

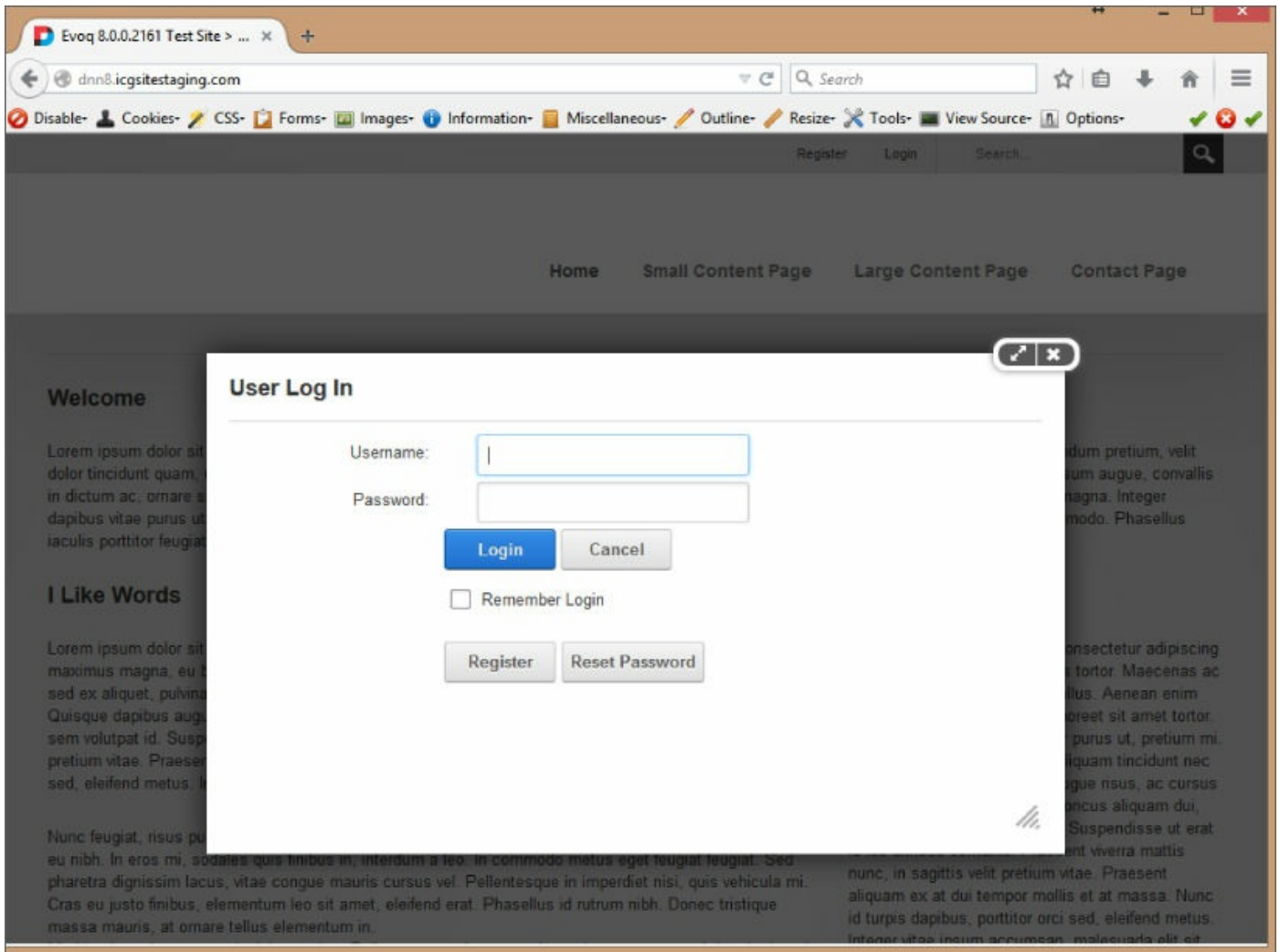


Figure 14.2

Modal dialogs can be used for any control other than the default module control. The process of instructing DNN that you want your module to open in a pop-up is as simple as supporting AJAX. [Listing 14.3](#) shows an updated snippet of the Guestbook module manifest that supports editing in a modal dialog.

[Listing 14.3](#): Guestbook Module Manifest Update with Modal Dialog

```
<moduleControl>
  <controlKey/>
  <controlSrc>DesktopModules/WROX/Guestbook/View.ascx</controlSrc>
  <supportsPartialRendering>True</supportsPartialRendering>
  <controlTitle/>
  <controlType>View</controlType>
  <viewOrder>0</viewOrder>
```

```
</moduleControl>
<moduleControl>
  <controlKey>Edit</controlKey>
  <controlSrc>DesktopModules/WROX/Guestbook/Edit.ascx</controlSrc>
  <supportsPartialRendering>False</supportsPartialRendering>
  <supportsPopUps>True</supportsPopUps>
  <controlTitle>Edit Content</controlTitle>
  <controlType>View</controlType>
  <viewOrder>0</viewOrder>
  <supportsPopUps>False</supportsPopUps>
</moduleControl>
```

By simply adding the `<supportsPopUps>` XML element with a value of `True`, you instruct DNN that you want to have your module display in this manner. When generating URLs to your additional controls using `ModuleContext`, DNN automatically adds the needed elements to open in a dialog.

It is important to note that although a module manifest might indicate that it supports pop-ups, it is not a guarantee that your control will be in a modal dialog. This is because site administrators can disable modal dialogs in the Site Settings area of their DNN install.

JavaScript, jQuery, and Custom Scripts

DNN as a platform has out-of-the-box support for jQuery and jQuery UI and has a centralized registration process for any third-party or custom JavaScript that you need to include in a module. It is important as a module developer that you leverage these extension points and APIs, as they help prevent multiple script references. They also tie into an optional process that can help site administrators minimize and combine script files to reduce overall page payload.

Using jQuery and jQuery UI

Many sections of DNN already have references to jQuery and jQuery UI, so it only makes sense that it's easy to ensure that jQuery or jQuery UI is included for your modules. [Listing 14.4](#) shows the options necessary to request registration of both jQuery and jQuery UI within a module's custom code.

[Listing 14.4](#): Request jQuery and jQueryUI Registration

```
//using DotNetNuke.Framework.JavascriptLibraries
JavaScript.RequestRegistration(CommonJs.jQuery);
JavaScript.RequestRegistration(CommonJs.jQueryUI);
JavaScript.RequestRegistration(CommonJs.jQueryFileUpload);
```

Using this method, a module can request DNN to register all of the available jQuery and jQuery UI features. If a module only needs jQuery, that is all you need to do. All registrations of this type will utilize preconfigured DNN references with regard to the jQuery and jQuery UI versions. Additionally, any configured jQuery CDN will be respected automatically. Manual addition, or referencing a different version of jQuery than the current DNN version, is strongly discouraged as conflicts often occur in these circumstances.

It is important to note when using these methods that they must be executed on *all* executions of a page and should not be executed inside of a check for postback. Additionally, not all of the jQuery UI CSS is included by default, so while the `RequestRegistration` method will ensure that the JavaScript is on the page, you will need to make sure to include any other CSS that your specific widgets require.

Including Knockout

DNN ships with Knockout and the Knockout Mapping script libraries. These are additional elements that can be leveraged by custom modules to support Single Page Applications (SPA) and other types of UI data binding. [Listing 14.5](#) shows the code necessary to include these libraries as part of a module.

[Listing 14.5](#): Request Knockout Registration

```
//using DotNetNuke.Framework.JavascriptLibraries;  
JavaScript.RequestRegistration(CommonJs.Knockout);  
JavaScript.RequestRegistration(CommonJs.KnockoutMapping);
```

You can see a common trend in the inclusion of JavaScript libraries. In DNN, a class called `CommonJs` is used to store references in the various included packages. Any and all libraries that exist in `CommonJs` should be referenced using this process to ensure that a custom module does not hinder the proper execution of a core DNN feature.

The `CommonJs` process provides only a code reference to a package name within the framework. As a developer, you may create your own packages for common libraries that you might wish to include. By including common script libraries as packages rather than embedded as part of your module, you might be able to help limit collisions and improve reusability across the modules.

Registering Custom Scripts and CSS

Developers who are new to DNN often use older methods to include a JavaScript file in a user control for user display. [Listing 14.6](#) shows a common practice that first-time developers use.

[Listing 14.6](#): Traditional JavaScript Registration

```
<script src="/DesktopModules/Guestbook/MyScript.js"/>
```

On the surface, this will probably work when only one instance of a module is added to a page. However, if a situation arises similar to the one illustrated at the beginning of the chapter, you'll need a method to limit the script

registrations to only once per page load, even when multiple modules request the same file.

This is where Client Resource Management comes in. It's a collection of objects inside the `DotNetNuke.Web.Client` assembly. Client Resource Management provides many features to the DNN Platform, including the management of script references for custom modules. This functionality is further enhanced by supporting actions such as minification and combination, if enabled by the host.

For module developers, it is recommended that Client Resource Management APIs be utilized for all script and CSS references to ensure the best compatibility with different environments. [Listing 14.7](#) shows an example of the code necessary to register one custom script as well as a custom stylesheet.

[Listing 14.7](#): Register Custom JavaScript

```
//using DotNetNuke.Web.Client.ClientResourceManagement;
ClientResourceManager.RegisterScript(
    this.Page,
    "/DesktopModules/Guestbook/myScript.js");
ClientResourceManager.RegisterStyleSheet(
    this.Page,
    "/DesktopModules/Guestbook/myStyle.css");
```

Using this process, the DNN APIs are now responsible for the proper handling of registration and will register the file only if it has not been registered in order to avoid duplication.

The example provided in [Listing 14.7](#) shows the most basic calls to that `RegisterScript` method. This works well; however, it should be noted that a few additional overloads give developers much greater control over the placement and priority of the individual referenced scripts. These additional overloads allow developers to specify a priority and provider, allowing for the most granular control of the script insertion.

Priority values are set using an integer value, and within the class `DotNetNuke.Web.Client.FileOrder` there is an enumeration showing the default values for all of the standard DNN configuration. The following table shows these default values. Items are processed in numerical order from

smallest to largest, and the default if not specified by anyone is a value of 100 to load after all other resources. This can be great if you need to do things in a particular place.

Name	Value
DefaultPriority	100
jQuery	5
jQueryMigrate	6
jQueryUI	10
DnnXml	15
DnnXmlJsParser	20
DnnXmlHttp	25
DnnXmlHttpJsXmlHttpRequest	30
DnnDomPositioning	35
DnnControls	40
DnnControlsLabelEdit	45
DnnModalPopup	50
HoverIntent	55

The second additional option allows you to specify a particular provider. The provider in this context allows you to control where the script will be injected inside of DNN. It is strongly recommended that you utilize this function to force items to use the `DnnFormBottomProvider` whenever possible to improve above-the-fold load times. The following table outlines the default providers included as part of DNN. For use within calls to `Register` you use the string representation of the name.

Provider Name	Functionality
<code>DnnBodyProvider</code>	Inserts the script to the body of the HTML document, at the top of the body.
<code>DnnCompositeFileProcessingProvider</code>	Ensures that the script goes through the composite file processing process.
<code>DnnFormBottomProvider</code>	Ensures that the script is added at the very bottom of the HTML form. This

	is the recommended option to use.
DnnPageHeaderProvider	Ensures that the script is registered in the page header.

DNN jQuery Plugins

In addition to the basic support for jQuery and jQuery UI, DNN offers a number of helpful jQuery plugins. These plugins have multiple benefits. They not only speed up the development process but assist in providing a consistent user experience. Registration of these plugins is completed similar to how you register the jQuery and jQuery UI libraries, using

```
CommonJs.DnnPlugins.
```

There are many controls exposed as part of this library; the following sections explore a few of the most common.

dnnAlert

The `dnnAlert` plugin is a jQuery-based replacement for the often-used basic JavaScript alert. A simple message displays to users with a single option. This is helpful when you want to display an error. Given the nature of this plugin, it should be called when you want an alert to be displayed; it's not attached to any particular element(s) in your module code. [Listing 14.8](#) shows a complete example of the `dnnAlert` plugin.

[Listing 14.8](#): dnnAlert Usage Example

```
$.dnnAlert({
  okText: '<%= Localization.GetString("OkButton.Text",
this.LocalResourceFile)%>',
  text: '<%= Localization.GetString("Message.Text",
this.LocalResourceFile) %>',
  dialogClass: 'dnnFormPopup myCustomDialog'
})
```

In this example, it is important to note that the `text` property is the only item necessary; the other two properties can be set if an override is needed. If you're overriding the `dialogClass` and not including `dnnFormPopup`, you will want to confirm that all the needed styles are applied.

dnnConfirm

It is common practice to require users to confirm actions that should not be triggered inadvertently. For example, users must confirm prior to deleting a record from the database or confirm before sending a bulk email. This is

where the `dnnConfirm` plugin comes in handy. By using this plugin with a simple jQuery selector, you can attach a confirmation process to any button or `LinkButton` without needing to manually add `OnClick` or `OnClientClick` handlers to the controls. [Listing 14.9](#) shows a sample implementation of a `dnnConfirm` dialog.

[Listing 14.9](#): dnnConfirm Usage Example

```
$('.addConfirmButton').dnnConfirm({
  text: 'Message to User Here',
  yesText: 'Yes',
  noText: 'No',
  title: 'Confirm Dialog Title',
  isButton: true
});
```

This listing uses the CSS selector `.addConfirmButton` to add a simple confirmation dialog to any controls with the specified class. With this single statement, users must select Yes from the dialog in order for the postback to occur.

dnnPanels

Another common user interface element found in DNN is the use of expanding panels of content. These are used in areas such as Site Settings, Module Settings, and other areas of DNN. These panels allow developers to group content into logical panels and then allow users to expand/collapse these items as they desire when working with the page. A key benefit to using `dnnPanels` over other options is that this control respects client-side validation routines and ensures that users will see errors, even on collapsed sections as they try to submit them. [Listing 14.10](#) shows example markup and JavaScript necessary to render a basic panel setup.

[Listing 14.10](#): dnnPanels Usage Example

```
<div id="dnnMyPanels">
  <h2 class="dnnFormSectionHead">
    <a href="">
      Panel 1
    </a>
  </h2>
```

```

<fieldset>
  <div>Panel 1 Content goes here</div>
</fieldset>
<h2 class="dnnFormSectionHead">
  <a href="#">
    Panel 2
  </a>
</h2>
<fieldset>
  <div>Panel 2 Content</div>
</fieldset>
<asp:hyperlink id="hlSubmit" runat="server"
  text="submit" cssclass="dnnPrimaryAction"/>
</div>
<script>
$('#dnnMyPanels').dnnPanels();
</script>

```

By dissecting the example in [Listing 14.10](#), you can learn a few key attributes about the `dnnPanels` plugin. The first item to note is that the content containing the panels is wrapped in a `div` with a unique ID. This provides an instruction point for the `dnnPanels` method. When initializing the panels, DNN will look at the HTML structure inside the `div`. From there, each panel follows a consistent structure. An `h2` with a CSS class for `dnnFormSectionHead` and a hyperlink comprise the panel heading. A `fieldset` directly after the `h2` is used to group the contents of the particular panel. The final item of importance is that the Submit button for the particular panels must be contained inside the main `div`. This ensures proper client-side validation.

For those looking to further customize the layout of the `dnnPanels` plugin, it is possible to override properties when calling `.dnnPanels()`. The following table lists the options, default values, and descriptions of these options.

Option	Default	Description
<code>clickToToggleSelector</code>	<code>h2.dnnFormSectionHead a</code>	The full selector for the panel heading. It's inside the <code>div</code> container that is initialized. The container should be expanded to include the <code>h2</code> and the <code>a</code> tag.
<code>sectionHeadSelector</code>	<code>.dnnFormSectionHead</code>	Defines the CSS class for the panel heading.

		panel.
regionToToggleSelector	fieldset	The select region that be expanded, upon selection
toggleClass	.dnnSectionExpanded	The CSS class added to the expanded
clickToToggleIsolatedSelector	a	The isolated element for toggle selection should trigger expand/collapse
validationTriggerSelector	.dnnPrimaryAction	Selector for controls that the client-validation
invalidItemSelector	.dnnFormError[style*="inline"]	The selector to identify an item after validation
validationGroup	' '	Ensures that the supplied validation is used when triggering validation
panelSeparatorSelector	.ui-tabs-panel	Identifies separator for individual
cookieDays	0	Determines how many days the panel selection should be

		Only applies if <code>cookieMill</code> is not specified.
<code>cookieMilliseconds</code>	1200000	Determines how many milliseconds a cookie session of users' panel selections will be valid. (Default is 1200000, or twenty minutes). Helpful for debugging postback.
<code>saveState</code>	true	Whether to save the expanded/collapsed state of the tabs in a cookie.
<code>defaultState</code>	first	The default state of the panel when the page loads. Can be 'first' (the first panel is expanded) or 'open' (all panels are expanded) or 'close' (all sections are collapsed).

If you use an advanced layout by overriding these properties, you will also most likely need to customize the CSS necessary to lay out the display.

dnnTabs

The final DNN jQuery plugin discussed in this chapter is the `dnnTabs` component. It's commonly used in complex setup screens such as Site Settings and Host Settings to provide tabbed organization for large amounts of content. [Listing 14.11](#) contains a detailed example of the minimum markup necessary to create a tab layout.

Listing 14.11: dnnTabs Usage Example

```
<div id="dnnMyTabs">
  <ul class="dnnAdminTabNav">
    <li><a href="#tabs-1">Tab 1</a></li>
    <li><a href="#tabs-2">Tab 2</a></li>
  </ul>

  <div id="tabs-1">
    Tab 1 Content
  </div>

  <div id="tabs-2">
    Tab 2 Content
  </div>
</div>

<script>
$('#dnnMyTabs').dnnTabs();
</script>
```

This control is even easier to work with than the `dnnPanels` control. All that is necessary is a wrapping element—in this example, the `div` with an ID of `dnnMyTabs`, an unordered list with anchor links, and a `div` element with an ID that matches the anchor link identifier. It is important when building your HTML and related elements to consider uniqueness, including situations where users might include more than one instance of your own module on the page.

If you want a more complex layout, there are a number of properties that you can use to customize the layout and validation routines, as shown in the following table.

Option	Default	Description
<code>validationTriggerSelector</code>	<code>.dnnPrimaryAction</code>	Selector for controls that trigger client-side validation.
<code>validationGroup</code>	<code>''</code>	Ensures that the supplied <code>validationGroup</code> is used when

		triggering validation.
invalidItemSelector	<code>.dnnFormError[style*="inline"]</code>	The selector to identify an invalid item after validation.
selected	-1	The index of the tab to select. With no tab is selected, the first tab would be selected.
disabled	false	true to disable tabs, or an array of zero-based indices of tabs that should be disabled; e.g., <code>[0, 2]</code> would disable the first and third tabs.
cookieDays	0	Determines for how many days a cookie storing the user's panel selection should be valid. Only applies if <code>cookieMilliseconds</code> is not specified.
cookieMilliseconds	1200000	Determines for how many milliseconds a cookie storing users' panel selections should be valid. (Default is twenty minutes.) Helpful for postback.

With the options available as part of the `dnnTabs` component, it is easy to combine them with other elements to persist user selections as well as integrate them with ASP.NET validation routines. That way, you can ensure that users see everything they should!

Handling Postbacks

In the previous examples, the initialization routines expect that the pages are always loaded fully. If you find yourself working in an environment where you have enabled support for AJAX updates, it's important to use an initialization strategy that not only initializes DNN jQuery plugins on initial load but also on future executions. [Listing 14.12](#) provides an example.

Listing 14.12: Initialization with AJAX Support

```
(function ($, Sys) {
    function setupMyPlugins() {
        /* Your initialization here */
    }

    $(document).ready(function () {

        setupMyPlugins(); //Run on document ready

Sys.WebForms.PageRequestManager.getInstance().add_endRequest(
        function (sender, args) {
            //Run on ajax endRequest
            setupMyPlugins();
        }
    );
});
}(jQuery, window.Sys));
```

In [Listing 14.12](#), any initialization routines such as `.dnnPanels` or `.dnnTabs` would be added to the line indicated by `Your initialization here`. The other portions of the script ensure that the setup routine is called when the first page loads as well as when any AJAX request is completed. This is accomplished by adding a function call when `endRequest` is raised by the `PageRequestManager`. `endRequest` will be called whenever *any* `UpdatePanel` postback completes, so you may want to inspect the arguments passed to the event handler to determine whether the postback applied to your module.

Implementing Consistent Design

Using a product such as DNN as a foundation for custom development has many advantages. One of these advantages is that a number of common elements exist as part of the application, and they allow developers to leverage UI styles that already exist in the application. By using these styles, which are included by default in any page of the application, you can create a consistent experience for users and visitors alike.

Messages and Alerts

After only a short period of interacting with DNN, you most likely have already encountered a few of the standard module message routines that produce messages such as “This is a warning,” “This is a success message,” “This is an error,” and so on.

Within your code-behind you can have DNN systematically insert a message with custom text using `DotNetNuke.UI.Skins.Skin.AddModuleMessage()` and the desired message text will be added to the top of your module. However, there are often times where you might want to include a message with this styling in other places in your modules, or possibly even in an HTML module somewhere. This is easy to do. [Listing 14.13](#) shows the code necessary for a green message window.

[Listing 14.13](#): Success Message HTML

```
<div class="dnnFormMessage dnnFormSuccess">  
Your message here  
</div>
```

Although [Listing 14.13](#) shows a success message, it is easy to adapt this example to the other message types. All messages rely on the common `dnnFormMessage` CSS class; the secondary class is used to set the color of the message. The following table lists the class names, color, and intended purpose for all available options.

Class	Color	Purpose
<code>dnnFormSuccess</code>	Green	Indicates a successful task completion. Examples include profile updated

		successfully, SQL query executed, or information saved.
dnnFormInfo	Blue	Used to provide additional information to the users. Great for providing guidance on how to complete a particular form.
dnnFormWarning	Yellow	Used to communicate a warning to the users. Great for information such as cautionary warnings or important information that should have attention drawn to it.
dnnFormValidationSummary	Red	Used to communicate an error condition to the users.

By utilizing these CSS classes and design patterns, your custom implementations can follow a similar workflow and communication pattern as the rest of the platform. This way, you provide users with a consistent interface that is easy to understand.

Forms and Buttons

Forms exist all over the DNN Platform, and skins must contain a set of styles used to control the layout of these elements. Following these standards will allow your modules to easily adapt to any changes a skin designer makes to these default styles. [Listing 14.14](#) shows a sample form implementation showcasing the various CSS classes in use.

[Listing 14.14: Sample Form](#)

```
<div class="dnnForm">
  <div class="dnnFormItem">
    <dnn:label runat="server" id="lblTest2" Text="Test 2"
Suffix=":"
    CssClass="dnnFormRequired"/>
    <asp:RadioButtonList runat="server" id="rblDemo"
    CssClass="dnnFormRadioButtons">
      <asp:ListItem Text="Item 1" Value="1"/>
    </asp:RadioButtonList>
  </div>
  <div class="dnnFormItem">
    <dnn:label runat="server" id="lblTest" Text="Test"
```

```

Suffix=":"
    CssClass="dnnFormRequired"/>
    <asp:TextBox runat="server" id="txtTest"/>
    <asp:RequiredFieldValidator runat="server" id="TestRequired"
        ControlToValidate="txtTest"
        CssClass="dnnFormMessage dnnFormError" text="Required"/>
</div>

<ul class="dnnActions dnnClear">
    <li><asp:LinkButton runat="Server" id="btnSave" text="Save"
        CssClass="dnnPrimaryAction"/></li>
    <li><a href="/" class="dnnSecondaryAction">Cancel</a></li>
</ul>
</div>

```

This is a very simple form layout with only a minimal number of CSS classes necessary to lay out the form. Depending on the skin used and the specifics of your layout, [Listing 14.14](#) will result in two fields being displayed with Save and Cancel buttons included at the bottom. Consider each of the CSS classes and their intended uses:

- `dnnForm`—This is the CSS class used to wrap the containing form. It ensures that all the needed styles can be applied.
- `dnnFormItem`—This is the CSS class that wraps an individual form item. You should have a label and control inside of this element.
- `dnnFormRequired`—This is a CSS class that can be applied to the `DNN LabelControl` and will result in a red * being displayed next to the label for required fields. (*Note:* In DNN 6.x, this CSS class was previously applied to the actual input element rather than the label. This design is *not* supported in DNN 7.x.)
- `dnnFormMessage dnnFormError`—This combination of CSS classes should be used with all ASP.NET validation controls to ensure proper display of validation errors to users.
- `dnnActions`—This is a CSS class applied to an unordered list to provide users with buttons for individual actions, such as Save and Cancel. Typically `dnnClear` is also used on this list to ensure correct alignment with the previous form items.
- `dnnPrimaryAction`—This is a CSS class applied to a button, `LinkButton`, or a regular HTML link that designates the control as the primary action a user should select. In the default skin, this will render the control as a blue

button with white text.

- `dnnSecondaryAction`—This is a CSS class applied to a button, `LinkButton`, or a regular HTML link that designates the control as a secondary action for a user. In the default skin, this will render the control as a light gray button with dark gray text.

Using Module CSS

When working with an interface design for a DNN module, it might be necessary to include a number of stylistic pieces that don't exist prior to starting. When you're looking for an easy way to include CSS as part of your module package and still want users to be able to customize the layout for their specific application, the `Module.css` file is a great asset.

This is an automatic feature of the DNN module system. You simply create a file in your module structure named `module.css` and include it as part of your package. It will be included as a reference anytime your module is listed on a page. DNN automatically handles the addition of the reference for you. No special action is needed by you as the developer.

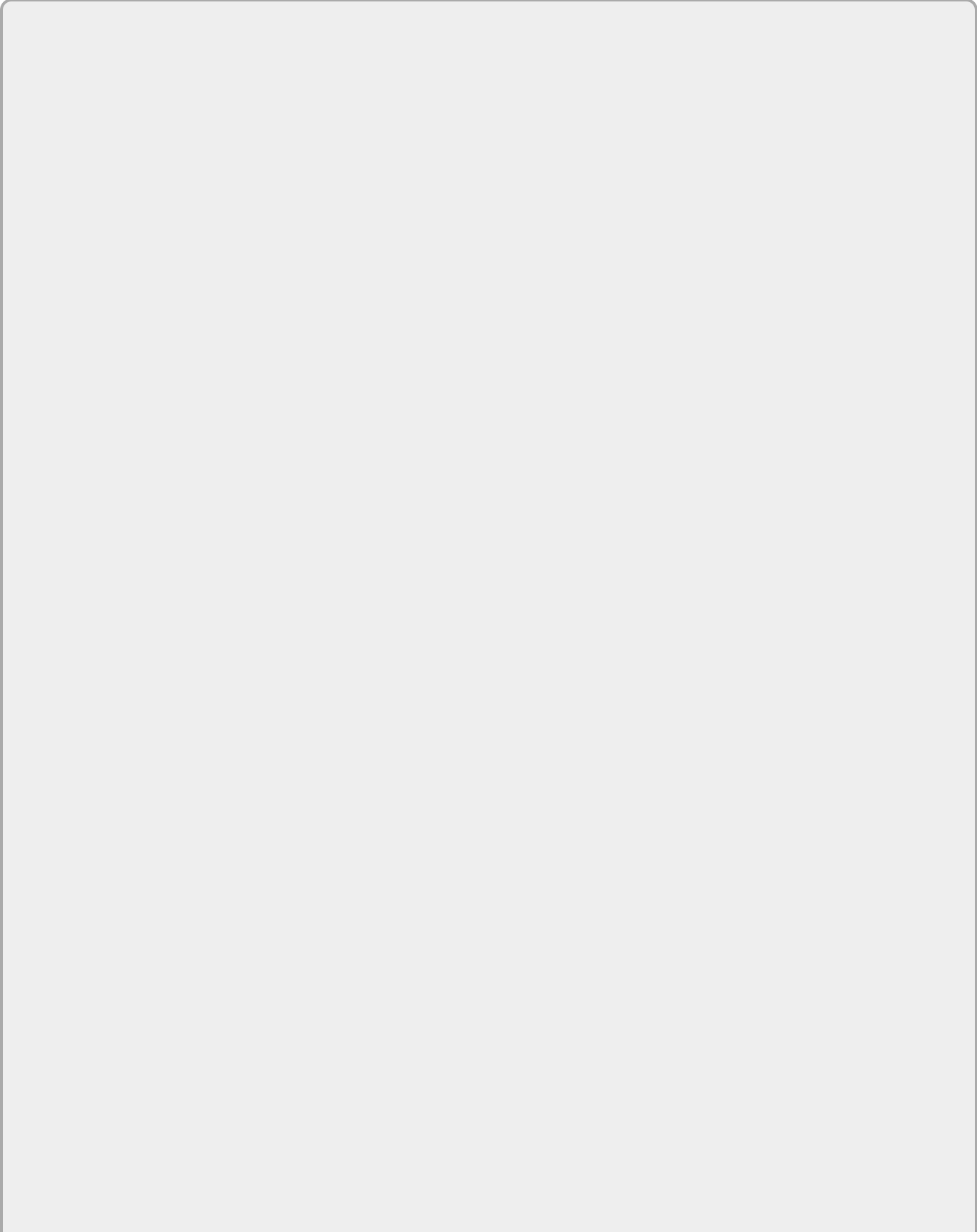
When you're using a module development template, this file is often included. To avoid adding unnecessary HTTP overhead to installed DNN sites if no CSS is included in this file, remove it from any distribution.

Summary

This chapter investigated the various tools and techniques that you can use to help create custom extension interfaces that are robust, consistent, and easy to build. When you leverage all of the tools, libraries, and components that DNN makes available, rapid development can become a reality.

Chapter 15

Developing Modules: Business Logic



What You Will Learn in This Chapter

- Navigating the DNN API
- Using common DNN controls
- Leveraging Web API
- Creating scheduled jobs
- Understanding DNN navigation options
- Building multiple UI controls

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 15](#) download and individually named according to the names throughout the chapter.

[Chapter 13](#) introduced a high-level overview of development in DNN and [Chapter 14](#) brought along discussions around the user interface. Now, it's time to dig in to the nuts and bolts of development and discuss the inner workings of the .NET side of development. This chapter investigates the API structure, shared controls, and more. The goal of this chapter is to expose you to the elements available to developers, leaving the specific implementation to the problem at hand.

Navigating with the DNN API

One of the most common complaints that arise from those new to working with DNN is that the API is cumbersome and hard to understand. Given the complex size of the API, this is understandable; however, armed with a few key points of understanding, it is much easier to navigate. To help dispel some of the confusion, let's dive into some often used namespaces in the DNN API to review their purpose and benefit to module developers.

DotNetNuke.Common Namespace

This namespace is often billed as one of the most complex to understand. That's because there is a lot of history as well as many amazing tools available. There is a complex structure of components nested under this namespace, and this chapter reviews the highlights of some of the most common elements used by developers.

DotNetNuke.Common.Globals Class

This is one of the most heavily used classes in the DNN Framework, so it's important that you understand its purpose. The `Globals` class provides access to information that is commonly needed across all bits of code. This class, although not marked as `static` in the code, contains only static members, so to use any function in this API, you simply use

`DotNetNuke.Common.Globals.METHOD`. [Table 15.1](#) highlights a number of the most common methods and their benefits to developers.

Table 15.1 Common Methods

Method	Purpose
<code>AccessDeniedURL()</code>	Returns an access denied URL that can be used to display a common access denied message to the user. An optional message can be passed, allowing customized display.
<code>ApplicationMapPath</code>	Provides access to the path on disk for the DNN installation. For example, <code>C:\inetpub\DNN</code> if this was the default installation location on a server.
<code>CleanFileName()</code>	Takes a filename and removes invalid characters from the string. Great for quick-and-easy management of user-supplied file uploads.

<code>ElapsedSinceAppStart</code>	Returns a timespan indicating how long the application has been active. This is great for diagnostic situations to identify warm vs. cold worker processes.
<code>NETFrameworkVersion</code>	Returns the current version number of the .NET Framework that the application is using. Great for understanding what is going on in the application.
<code>IsEditMode</code>	A great way to check if the current user is viewing the site in “edit mode.” This enables you to add/remove features/functions depending on the user's current mode.
<code>LoginURL()</code>	Returns the URL for the login page. You may pass an optional <code>ReturnUrl</code> path to this method to ensure that users return after login. Great for providing users a detailed option to log in and return to your application regardless of the host configuration.
<code>NavigateURL()</code>	The key for generating links from page to page. Later in this chapter, the section entitled “Managing Navigation” discusses this function and a few related functions in detail.
<code>QueryStringEncode()</code>	A very handy method for encoding a value to be placed into the querystring. Often used in conjunction with various link-building processes.
<code>QueryStringDecode()</code>	The paired method to decode any URL-encoded values that might have been created with <code>QueryStringEncode()</code> .

At a high level, most projects will use the `Globals` class for one or more functions. Be sure to check out the other properties that are exposed.

DotNetNuke.Common.Lists.ListController

DNN contains a list-management feature, found under `Host` ⇨ `Lists` and `Admin` ⇨ `Lists`, that allows users to manage lists of information. Out of the box, DNN comes with lists for Countries, Regions, and Currencies, among others. [Listing 15.1](#) shows how the list controller can be used to extract a list of states and populate a drop-down list with the values.

[Listing 15.1](#): Using ListController to Bind a List of States

```
var controller = new DotNetNuke.Common.Lists.ListController();
var states = controller.GetListEntryInfoItems("Region",
"Country.US");
ddlStates.DataSource = states;
ddlStates.DataValueField = "EntryID";
ddlStates.DataTextField = "Text";
ddlStates.DataBind();
```

In the previous code, `GetListEntryInfoItems` gets two values. The first value is the list you are requesting, which in this case is `Region`, and the second is the parent list that you want to use to locate the region (in this case, it's `Country.US`). This nested architecture and default functionality can aid developers in ensuring that information is centrally managed and easy for users to find and use.

DotNetNuke.Common.Utilities.CBO Object

The `CBO`, or Common Business Object, is a key portion of the DAL and DAL+ data access strategies that were introduced as part of [Chapter 13](#). Static helper methods, such as `FillObject<T>` and `FillCollection<T>`, take an `IDataReader` and fill single objects or collections of objects.

DotNetNuke.Common.Utilities.DataCache Object

An integral portion of any application development framework is a robust caching process. DNN is no different in this regard, and the `DataCache` object is the gateway for properly storing information into cache for later use. This cache store is an application-level caching implementation, which means that the information that is stored in the cache system is application specific, not user specific. [Listing 15.2](#) includes three examples of adding an item to the cache.

[Listing 15.2](#): Storing Items to the Cache

```
//Using DotNetNuke.Common.Utilities at top
//Default duration
DataCache.SetCache("WroxDemoKey", "Test");
//Sliding Expiration
DataCache.SetCache("WroxDemoKey", "Test",
```

```
TimeSpan.FromMinutes(30));  
//Absolute Expiration  
DataCache.SetCache("WroxDemoKey", "Test",  
DateTime.Now.AddMinutes(45));
```

The first example is the most basic. It provides a string-based cache key and the item to be cached. Defaults are used for expiration time and policy. However, you can have more granular control over the cache expiration policy. The second example instructs the cache system to utilize a sliding cache expiration process with a 30-minute window for removal from the cache. The third example shows a cached item where a specific expiration `DateTime` is used. These options provide developers with robust options for determining how long information should persist.

Retrieving information from the data cache is also a simple process. [Listing 15.3](#) provides a basic example of retrieving information from the cache.

[Listing 15.3](#): Getting an Item from the Cache

```
//using DotNetNuke.Common.Utilities; at top  
var args = new CacheItemArgs("WroxDemoKey");  
var lowPriorityArgs = new CacheItemArgs("WroxDemoKey",  
CacheItemPriority.Low);  
var timeoutArgs = new CacheItemArgs("WroxDemoKey", 20);  
var foundDate = DataCache.GetCachedData<string>(args, e =>  
RetrieveAndStoreContent());
```

It is important when using these `GetCache` methods that a null check is completed, as items can be removed from the cache system for a number of reasons. Items might be removed from the cache due to the duration of time they have been in the cache or if the cache system is resource constrained.

DotNetNuke.Entities Namespace

This namespace is an important one to remember. This is the storage location of all of the key entities used by the application. Information such as users, sites, pages and similar functions are found in this namespace.

In a perfect world, we could cover every single namespace and entity in this important namespace. However, with this namespace it is not entirely possible, as the `Entities` namespace holds a majority of the objects used by

DNN and additions are often made release to release. It's more important to discuss the similarities and to help you gain an understanding of the namespace in general.

_Info Objects

When you begin reviewing the various sub-namespaces under the `Entities` namespace, items such as `UserInfo`, `RoleInfo`, and `TabInfo` start to surface. These elements are the classes that hold data for the particular entity, and some might call them models or data transfer objects, although they are a bit more complex than any general description can provide.

These objects have no static members associated with them and typically represent a single element of the particular type. These objects usually contain *only* properties and not actions or methods. Because of this, we'll move on to the next most common object structure, the controller object.

_Controller Objects

Typically found in pairs with `_Info` objects, `_Controller` objects provide the actions and processes to interact with objects. For example, the `UserInfo` class represents a single user, and the `UserController` can interact with these objects. Many of these controllers have a long-running API history that adds to their confusing landscape.

At a high level, any time you want to interact with a controller object, look for your desired function as a static implementation, such as

`UserController.GetUserByEmail("test@wrox.com")` rather than create an instance of the user controller. Older versions of DNN supported some instance methods as well as some static implementations. [Figure 15.1](#) shows the IntelliSense of the `UserController` when working with a concrete implementation.

```
O.references
public partial class View : PortalModuleBase, IActionable
{
    O.references
    protected void Page_load(object sender, EventArgs e)
    {
        var test = new UserController();
        test.
    }
}

public
    AddUser (UserInfo objUser):int
    DeleteAllUsers
    DeleteUser
    DeleteUsers
    DisplayFormat
    Equals
    FillUserInfo
    GetCacheKey
    GetHashCode
    GetSuperUsers
    GetType
    IActionable
}
```

Obsolete: Deprecated in DNN 5.1. This function has been replaced by UserController.CreateUser

(UserInfo objUser; bool addToMembershipProvider):int

Figure 15.1

Figure 15.1 shows that almost every method on the concrete implementation side is marked as “obsolete.” For those getting started with DNN, it is *very* important to avoid these methods. Also note that older documentation around DNN development might reference a method that shouldn't be used.



NOTE

Most DNN controllers use `UserController.Instance` nowadays. It gives you the most up-to-date API, over the static methods on `UserController`, in addition to supporting inversion of control. It provides a static `Instance` property, which gets the current singleton instance behind an interface (it also provides a way to fake the instance in tests). Examples include `UserController.Instance`, `CBO.Instance`, `EventLogController.Instance`, `ModuleController.Instance`, and `TabController.Instance`.

Host and PortalSettings Objects

The only entities that require special attention in the `Entities` namespace are the `Host` and `PortalSettings` objects. These classes are extremely helpful as they provide access to the site and host settings collections as standard objects, rather than a `HashTable`. Properly cached and accessible from the DNN, many pieces of information are available here that can be helpful for developers on a daily basis. The `Host` object can be found under `DotNetNuke.Entities.Host.Host`, and the `PortalSettings` object is found under `DotNetNuke.Entities.Portals.PortalSettings`. The `Current` static property of `PortalSettings` provides the site information for the current request.

[Table 15.2](#) shows some helpful properties of the `Host` object and [Table 15.3](#) outlines helpful properties associated with the `PortalSettings` object.

[Table 15.2](#) Helpful Host Object Properties

Property	Usage
<code>AllowedExtensionWhitelist</code>	This property exposes an instance of the <code>FileExtensionWhitelist</code> , allowing an extension to be quickly validated against the host extension file types. This property helps you adhere to configured settings for user uploads.
<code>DebugMode</code>	This Boolean flag can be changed by the site owner under the Host Settings page, and modules can use this to enable debug

	functionality in their extensions to assist with diagnostics.
HostEmail	The email address configured for the host. This value is beneficial when you're sending system notifications.
GUID	This is a Globally Unique Identifier for the installation of DNN. Many module developers will utilize this GUID to identify the installation for licensing purposes.
HostTitle	The title of the host as defined by the user.
HostURL	The configured URL for the host.

Table 15.3 Helpful PortalSettings Properties

Property	Usage
ActiveTab	Provides the <code>TabInfo</code> for the current page being requested.
AdministratorId	Provides the ID of the site administrator's user account.
AdministratorRoleId	Provides the ID of the role used to define users as administrators of the portal. A secondary <code>AdministratorRoleName</code> property exists to provide access to the name.
Description	The title of the site as supplied under Site Settings.
Email	The administrator email for the portal. When sending site-specific emails, it's best to use this address.
ErrorPage404	If the user has configured a custom 404 error (not found) page, the ID of this page will be provided by this property. If not, <code>Null.NullInteger</code> (i.e., -1) will be provided.
GUID	This is a Globally Unique Identifier for this site within the installation. Many module developers use this GUID to identify a site or sites for per-site licensing schemes.
HomeDirectory	This provides the relative path to the home directory

	for the site. By default, it's <code>/Portals/{PortalId}</code> , where <code>{PortalId}</code> is the site's ID, but users can change it. Using this value is highly recommended to provide the most robust support.
<code>HomeDirectoryMapPath</code>	This provides the full path, on disk, to the home directory.
<code>HomeTabId</code>	This provides the ID of the page that has been defined as the home page of the site.
<code>PortalAlias</code>	This provides a <code>PortalAliasInfo</code> for the site alias used by the current request. Great for linking situations where other options might not be as feasible.
<code>RegistrationTabId</code>	If configured by the user, returns the ID of the custom user registration page as defined in Admin Site Settings.

DotNetNuke.Framework Namespace

This namespace contains a number of key functions that have been discussed in earlier chapters. However, this is the key to the “framework” elements that comprise of the DNN solution. As such, there are a few key items to be aware of here.

AJAX Class

This class provides a number of static methods and options. You can use this class to request an AJAX Script Manager and use it to add scripts to the management process. One of the most common methods is `AJAX.RegisterPostBackControl()`, which allows you to define controls within your view that should trigger a full postback to the server rather than an AJAX partial update.

CDefault Class

One of the most often-overlooked classes in the DNN Framework is `CDefault`. This is the backing code file that represents the `default.aspx` file that's contained as part of the DNN installation. This is the base page for all rendered content as part of DNN. Using a simple casting process to convert `this.Page` to `CDefault` opens up a world of options. [Listing 15.4](#) shows a few

examples of what you can do using the `CDefault` class.

Listing 15.4: Updating Page Details

```
//Cast page to DNN page
var thePage = (CDefault) Page;

//Update values
thePage.Description = "Force change";
thePage.Title = "Set new title";
thePage.KeyWords = "Set new Description";

//Add controls
var toAdd = new HtmlLink();
toAdd.Href = "MyFile.css";
toAdd.Attributes["rel"] = "StyleSheet";
thePage.Header.Controls.Add(toAdd);
```

Given that `CDefault` is the base class that represents any page, it has a number of properties that are helpful to developers. The most popular of these are the `Description`, `Title`, and `Keywords` properties. If a custom module sets a value for one of these properties, you can update the defaults for the currently rendered page. If you have modules with dynamic content concerns, this is a must-use feature.

[Listing 15.4](#) also shows that `CDefault` gives module developers access to manipulate the document header by adding new controls. Although this example shows the addition of a stylesheet, it illustrates the process that you could follow for other types of content that would be more appropriate.

PageBase Class

As a DNN developer, most of the time you'll want to be working fully inside the context of the DNN page lifecycle. This means building your user interfaces as User Controls and running them alongside all other items. However, in certain circumstances, it might be more appropriate to build a standalone `.aspx` page. In these situations, it is helpful to use the `PageBase` class as your base class. This will provide better access to DNN integration information such as the `PortalAlias` and related data.

DotNetNuke.Security Namespace

Earlier in this chapter, when you looked at the `DotNetNuke.Entities` namespace, you most likely thought something was missing. Well, there was! All objects related to security in the DNN Framework, although associated with the object in the `Entities` namespace, live off on their own namespace. This is one of the biggest areas of confusion for individuals new to the DNN platform.

Given the unique nature of the objects and classes contained in this large namespace, this chapter focuses on a few specific sub-namespaces. The goal is to help guide you down the correct path of security-related functionality.

Membership Namespace

In DNN, users are entity objects defined in the `DNN.Entities` namespace. Users refer to the name, display name, and other profile style attributes associated with an account. The DNN system uses a concept of membership to define the actual authentication/login information.

At a high level, the DNN Membership subsystem that is defined in this namespace relates to the functionality that comes from the traditional ASP.NET membership system (`aspnet_users` and `aspnet_membership` database tables) that experienced .NET developers have encountered elsewhere. By utilizing this proven foundation, DNN has a solid base to rely on.

It is important to note that a user entity is tied to its membership entity through an association based on the username. This is the joining process between the `aspnet_tables` and the `Users` table that is part of DNN. All DNN functionality is then later tied together using the integer-based ID that is assigned to the `Users` table.

As a developer, you will use methods in the `Membership` namespace for setting, storing, and resetting user accounts. Additionally, it is a foundational element of a user's entire profile.

Permissions Namespace

The `Permissions` namespace is not used frequently by module developers, but it is important to be aware of its existence. The classes in this namespace define the default permissions sets and behaviors in the system. For example, this includes page, module, folder, and file permissions. This is an extensible area of DNN. You can see examples of the extensibility of this area by looking at the feature definition between the DNN Platform product discussed in this

book and the Evoq product lines.

Roles Namespace

As a DNN developer, you'll likely use the `Roles` namespace on a regular basis. Roles work just like the other entities do in the `Entities` namespace. Using a controller with static methods is the primary way to work with Role entities. You have the ability to query the system for roles, users in roles, and roles of users.

One best practice worth recommending here is that the roles system in DNN is dynamic. Users should have the ability to add/remove/delete roles. You'll need to use `UserInfo.IsInRole("RoleName")` to see if a user is a member of a role first. When storing a role in your module, it is most important to store the role ID value rather than the role name. If you take this small step of preparation, you will ensure that your module can stand the test of time, and the crazy site administrator! Future-proofing your modules for the actions that a user might take is what sets the good module developers apart from the mediocre ones.

DotNetNuke.Services Namespace

Looking for a broad generalization of the `Services` namespace, it consists of the stuff that has nowhere else to go. You could argue that many of these items could be in the `Common` namespace, and some developers have even logged suggestions that these should be in the `Framework` namespace.

Regardless of their placement, the elements in this namespace are elements that DNN makes available as “services” to those developing on top of the framework. These are not key UI elements or foundational building blocks to base your solutions on. These are the things that make your life easier as a module developer. They save you time.

As the chapter did with the other namespaces that are chock full of classes, this section investigates only the key common services found here.

Exceptions Class

In a perfect world, you would have code that worked flawlessly all of the time and nothing would ever go wrong. Realistically, that just isn't a reality that can be easily achieved. As such, DNN provides a bunch of helpful bits to help manage those exceptions. The key is to log the exceptions so that they can be

properly diagnosed later but to *not* show those exceptions to public users who might maliciously use information in detailed errors.

[Listing 15.5](#) shows a few options available to developers once an exception occurs. This example is manually creating an `AccessViolationException`; however, any exception would work.

[Listing 15.5](#): Getting an Item from the Cache

```
var myException = new AccessViolationException("Invalid Access");
//Log as a normal error
Exceptions.LogException(myException);
//Process as a module error
Exceptions.ProcessModuleLoadException(this, myException);
Exceptions.ProcessModuleLoadException("Oops! Try again", this,
myException);
```

The most basic example `LogException` does exactly what its name says: it logs an exception. That means it simply records the exception in the `EventViewer`. The `LogException` method is silent on the user's end and is your best bet if there is no need to communicate to the user that something has gone wrong.

The second example, `ProcessModuleLoadException`, is a more robust DNN method. It internally logs the exception and takes in the current control for contextual reference. With this information, it can add a standard DNN error message with a note to the user that an error has occurred. It handles the proper security display for messaging as well. If the user is a host user, all the details about the exception are shown on the page. If it's any other type of user, the generic "An error has occurred" message is displayed.

The third example is `ProcessModuleLoadException`, where in addition to providing the context and exception information, a "human-friendly" error message appears. In these cases, the human-friendly message is displayed to the user.

Whether you need to silently handle an error or display that you are experiencing issues, you have methods available to you in the `Exceptions` class. It should be noted that there is also the `ProcessPageLoadException` exception, which has a similar purpose; however, it shows the error at the page level not at the module level. It's best for overly critical failures.

Mail

Email is a part of our daily life, and it should be no different in the context of DNN. As with the `Exception` class, email is very easy to distribute as a module developer in DNN. [Listing 15.6](#) shows a simple example of email distribution using the `Mail` class method, `SendEmail`.

[Listing 15.6](#): Sending Email

```
var to = "test@test.com";  
var from = "test@from.com";  
var subject = "Test Email";  
var body = "<p>This is a test!</p>";  
DotNetNuke.Services.Mail.Mail.SendEmail(from, to, subject, body);
```

As you are looking at the API, it seems very simple. It's important to notice that there is a legacy `SendMail` method that requires you to supply SMTP server information. This newer method should be used instead. When you use the simple overload that takes the distribution information, all other information is obtained for you. This allows site administrators that are using “per-portal” SMTP configurations to use the same email distribution channels than those with the older host-only configurations are using.



NOTE

Be sure that your “from” addresses match your users' expectations. SMTP servers often have limited allowable senders. Users will often configure this to be the `Host.Email` or `PortalSettings.Email` value for the installation and/or site.

Using Common DotNetNuke Controls

There are a number of controls that allow DNN developers to reuse components so as to provide a consistent look and feel to their applications. This section explores the usage patterns of a few of these components.

LabelControl—Field Inputs

This control is one of the most commonly recognized. The field label has localizable text and help content. [Figure 15.2](#) shows this control as seen from the Settings view of a default HTML module in the DNN interface.

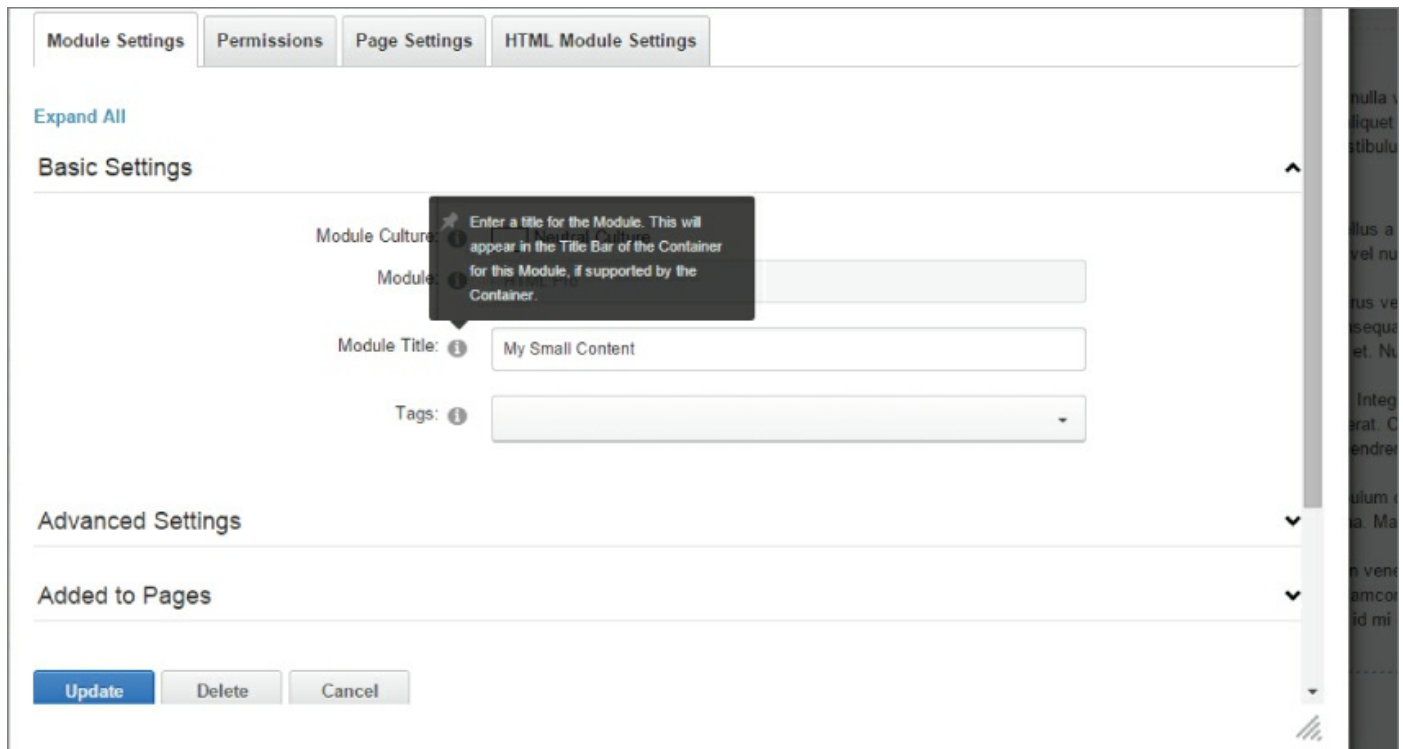


Figure 15.2

Using this control is a simple two-part process. The first step is to ensure that the top of your User Control has a `Register` tag that defines the association and references the control source. Once this is completed, you can use the newly registered control as many times as you need in your User Control. [Listing 15.7](#) shows the needed markup for the `.ascx` file of a module control in order to add the reference and provide one instance of the control on a page.

[Listing 15.7](#): Using the Label Control

```
<%@ Register tagPrefix="DNN" tagName="Label"
    src="~/Controls/LabelControl.ascx" %>

<dnn:label id="lblHeading" runat="server" suffix=":"
    ControlName="txtHeading"/>
<asp:TextBox runat="server" ID="txtHeading"/>
```

Visual Studio may provide a warning regarding the `src`. This is a message that can be ignored; when the code is running within the context of DNN, it will be a non-issue. Otherwise, a few other items are worth mentioning. The suffix is a value added to the end of the prompt text and the `ControlName` property is used to associate the label with an input. If you're concerned about Section 508 compliancy (for those with low/limited vision), it is imperative that every label be associated with its appropriate input using this `ControlName` option.

Note that there are no values supplied for the `Text` or `HelpText` properties of the control. This is because the default localization will pull the values from the current control's `.resx` file. If you provided two values in the `.resx`, one for `lblHeading.Text` and one for `lblHeading.HelpText`, the values will automatically fill in the label.

TextEditor—Rich Text Editing

When you're working in a CMS, you will frequently encounter an input for some sort of information that needs to be displayed to the users. In many cases, you'll need to collect more complex information from the user than just static, unstyled text. That's where the `TextEditor` control, which is part of the DNN core, comes in. [Figure 15.3](#) shows a default view of the editor interface that is created when you use this common control.

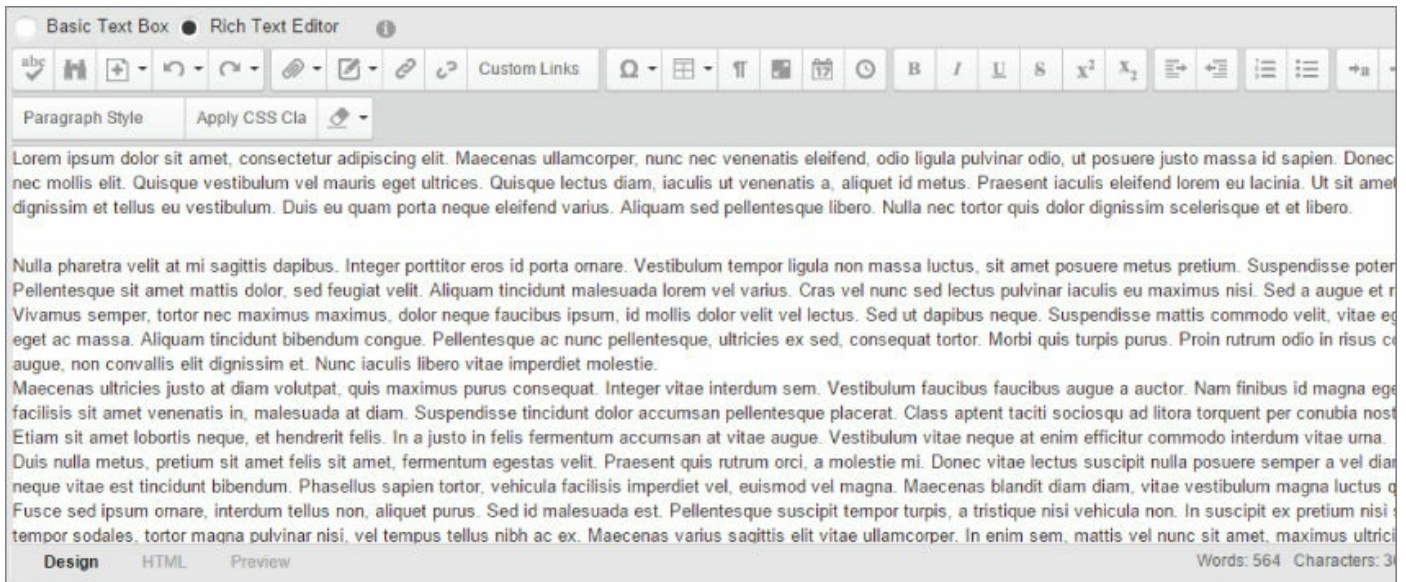


Figure 15.3

This is the common full HTML editor that users have come to expect. By utilizing this control, you can offer your users an experience that they are accustomed to. By using the `TextEditor` control as well, you can ensure that site administrators can use the default DNN functionality to change toolbars, layouts, and even file upload options. This is all done without you needing to make any changes to your module. [Listing 15.8](#) shows the code necessary to utilize the `TextEditor` control.

[Listing 15.8: Using the TextEditor Control](#)

```
<%@ Register tagPrefix="DNN" tagName="TextEditor"
    src="~/Controls/TextEditor.ascx" %>

<dnn:TextEditor id="txtInput" runat="server" HtmlEncode="false"/>
```

Just as with the sample of code used as part of the `LabelControl` demo, this example has a single register declaration and an individual usage of the control. In [Listing 15.8](#), the optional parameter for `HtmlEncode` is set to `false`. This ensures that when the data from the user is retrieved via the `.Text` property of the control, it's formatted as input text and not as HTML encoded. Use this setting at your discretion for proper retention and display of user inputs.

When working with Visual Studio, it is often the case that it will not properly declare the backend property with the `TextEditor` class. As such, in your

`.ascx` for each of these controls you will want to declare a property explicitly such as `protected TextEditor txtInput;` for the control defined in [Listing 15.8](#). This is necessary as Visual Studio by default will create the control as a simple `UserControl` and not the robust `TextEditor` input. If you do this after you declare the control in your `.ascx`, you might need to manually remove the declaration from the `.designer` file.

Leveraging Web API

One of the most popular aspects of modern web development is the ability to build rich user interfaces and the increasing popularity of the *single page application*. A key component to this is the ASP.NET Web API framework, which allows developers to quickly and easily create web services.

Although it's a great product, Web API on its own conflicts with the generic and ever-expanding model that is DNN development. As such, DNN has its own implementation of Web API extensions that allow users to integrate their own solutions and to tightly tie into the DNN security model.

Implementation of a DNN Web API solution is completed using a simple three-step process, described in the following sections.

Adding the Proper References

The first step to using DNN's implementation of Web API is to add references to all needed assemblies. The fastest method to complete this task is to add the `DotNetNuke.WebApi` NuGet package to your solution. A simple one-line call in the NuGet Package Manager console you will set you up for success.

If you want to add the references manually, you need to add the following to your solution:

- `DotNetNuke.Web.dll`
- `System.Net.Http.dll`
- `System.Net.Http.Formatting.dll`
- `System.Web.Http.dll`

With all of these references added to your solution, you're ready to move on to the next step of the process, which is building the `RouteMapper`. The `RouteMapper` defines the route that your API calls respond to.

Defining the Web API RouteMapper

Just as with a standard implementation of Web API, it is necessary to define the route, or method, by which users will communicate with your services. You do this by creating a class that implements the `IServiceRouteMapper` interface, which is part of the `DotNetNuke.Web.Api` namespace.

[Listing 15.9](#) shows a solution that creates a `WroxDemoRouteMapper` that defines

a single HTTP route.

Listing 15.9: Creation of RouteMapper

```
public class WroxDemoRouteMapper : IServiceRouteMapper
{
    public void RegisterRoutes(IMapRoute mapRouteManager)
    {
        //Map default route, will result in responding to
        // /DesktopModules/WroxDemo/Api/{controller}/{action}
        mapRouteManager.MapHttpRoute
            ("WroxDemo", "default", "{controller}/{action}",
            new[] { "Wrox.Api.Controllers" });
    }
}
```

There are a few key points you need to understand here. The name of the class itself does not matter; the key is the implementation of the `IServiceRouteMapper` interface. When implementing this interface, you use the single method called `RegisterRoutes`. At this point in time, you create your desired route for your API. It is important to look at this process with great detail; [Table 15.4](#) describes each parameter in more detail for mapping the `HttpRoute`.

Table 15.4 RegisterRoutes Method Parameters

Parameter	Purpose
"WroxDemo"	Provides the unique path for the route. It is added after <code>/DesktopModules/</code> to start the route identification. The best recommendation is to use your module folder path because it avoids collisions.
"Default"	A name for the route. If needed, you can create more than one route, for example, when you need to support complex route paths. This value must be unique in combination with the path value.
"{controller}/{action}"	Defines the key metrics of the route, for the information that comes after the start of the route endpoint. It defines two replacements, <code>Controller</code> and <code>Action</code> , which will map to the soon-to-be-

	<code>created Controller.</code>
<pre>new[] { "Wrox.ApiControllers" }</pre>	<p>Defines the individual namespaces that will be probed to identify controllers that are served as part of this route definition. Multiple namespaces can be included here. Do <i>not</i> provide fully qualified class names, only namespaces.</p>

So, what does this do for you? With this snippet of code, if you install the module to your DNN installation, it will be set up to start processing API calls with an API base URL of `/DesktopModules/WroxDemo/Api/`. You have not yet defined any controllers, so you cannot test it quite yet.

One common misconception for those working with `RouteMapper` for the first time is that it seems too simple. It is this simple; there is no need for extensive configuration or tedious information management!

Creating the Controllers

The final step, and the most critical, is defining your controller. The `RouteMapper` tells the requests where to go. The controller is the brains of the operation and is responsible for the final execution of all requests. When you create a controller, the rules for creation are quite simple. First, the controller must inherit from the `DnnApiController` base class. Second, its name should end with `Controller`, and lastly, it needs to be located in one of the namespaces identified by the `RouteMapper`. From there, DNN will interpret the controller and automatically map it and the actions (methods) to the defined route(s).

[Listing 15.10](#) shows a complete example of a controller with two distinct methods.

Listing 15.10: GuestbookController Example

```
namespace Wrox.ApiControllers
{
    public class UserQueryResult
    {
        public bool WasSuccessful { get; set; }
        public string ErrorMessage { get; set; }
        public string EmailAddress { get; set; }
        public int UserId { get; set; }
        public string Username { get; set; }
    }
}
```

```

    }

    [SupportedModules("Wrox.Guestbook")]
    public class GuestbookController : DnnApiController
    {

        public HttpResponseMessage GetAllEntries(int moduleId)
        {
            return this.Request.CreateResponse(HttpStatusCode.OK,
                MyDatabase.GetAllEntries(moduleId));
        }

        [DnnModuleAuthorize(AccessLevel =
SecurityAccessLevel.Edit)]
        public HttpResponseMessage GetUserDetails(int portalId,
int userId)
        {
            var user = UserController.GetUserById(portalId,
userId);
            if (user != null)
            {
                var result = new UserQueryResult
                {
                    EmailAddress = user.Email,
                    UserId = user.UserID,
                    Username = user.Username,
                    WasSuccessful = true
                };

                return
this.Request.CreateResponse(HttpStatusCode.OK, result);
            }
            else
            {
                var result = new UserQueryResult
                {
                    WasSuccessful = false,
                    ErrorMessage = "Unable to locate
user"
                };

                return
this.Request.CreateResponse(HttpStatusCode.OK, result);
            }
        }
    }
}

```

Using the known base route path from before—with a controller name of `Guestbook` and a `GetAllEntries` action—you can identify the proper URL needed to call the first method:

/DesktopModules/WroxDemo/Api/Guestbook/GetAllEntries?moduleId=1.

Just as with standard ASP.NET routing, it does not have any additional route parameters because anything that needs to be mapped as a parameter to the method call will be included on the end as a querystring. The result will be an XML or JSON response based on the accept header provided by the caller. Nothing else needs to be done on your part for this all to come together nicely.

The second method is worth calling attention to, as it provides for two additional steps. It requires that the user calling the API be authenticated and have edit rights to the module associated with the call. It's easy to secure access to an action by using the `DnnModuleAuthorize` attribute with an access level or permission key. The actions of this secondary API call are similar to that of the other call.

In order to associate a request with a module and use its permissions, the request needs to include `TabId` and `ModuleId` parameters. The easiest way to include these is to use the `$.ServicesFramework` JavaScript API (call `DotNetNuke.Framework.ServicesFramework.Instance.RequestAjaxScriptSupport` to ensure the API is available). [Listing 15.11](#) shows an example of making a service call from a module.

Listing 15.11: GuestbookController AJAX Call

```
<script>
  $(function () {
    var sf = $.ServicesFramework(<%: ModuleContext.ModuleId %>);
    $.ajax({
      url: sf.getServiceRoot("WroxDemo") +
'Guestbook/GetAllEntries',
      data: { moduleId: <%: ModuleContext.ModuleId %> },
      beforeSend: sf.setModuleHeaders,
      success: function (response) {
        console.log(response);
      }
    });
  });
</script>
```

Using DNN's Web API implementation, you can easily add services to your deployment while still remaining highly integrated with DNN.

Creating DNN Scheduled Jobs

Another commonly misunderstood process is creating a scheduled job in the DNN environment. Scheduled jobs are a great way to offload processing that doesn't need to be interactive. They allow tasks to be done away from the UI threads, which helps the users. [Listing 15.12](#) shows a simple example.

Listing 15.12: Sample Scheduled Job

```
public class DemoJob : SchedulerClient
{
    public DemoJob(ScheduleHistoryItem oItem)
    {
        ScheduleHistoryItem = oItem;
    }
    public override void DoWork()
    {
        try
        {
            //Perform required items for logging
            Progressing();

            //Call our process method
            CustomBusinessLogic();


            //Show success
            ScheduleHistoryItem.Succeeded = true;
        }
        catch (Exception ex)
        {
            ScheduleHistoryItem.Succeeded = false;
            InsertLogNote("Exception= " + ex);
            Errored(ref ex);
            Exceptions.LogException(ex);
        }
    }

    private void CustomBusinessLogic()
    {
        //Do Actual Work Here
    }

    private void InsertLogNote(string message)
    {
        //Helper method to make adding to the log easier
        ScheduleHistoryItem.AddLogNote(message + "<br/>");
    }
}
```

The overall structure of a scheduled job is simple. However, there are a few common gotcha elements to address. First of all, scheduled tasks inherit from the base `SchedulerClient` class. To be able to execute successfully, as well as to report status, scheduled jobs must have a single constructor that accepts and stores a `ScheduleHistoryItem` and updates the internal `ScheduleHistoryItem` property. Failing to do this will result in a job that not only will not execute but also won't provide any feedback.

The secondary requirement of all scheduled jobs is the existence of a public override `DoWork()` method. This method is called for each execution. You can see that you have a specific flow, with a global `try/catch`. This global exception handling is *required*. Without it, any exception could manage to go fully unnoticed, and the scheduled job would not record itself as complete. At the bottom of this class is a helper method called `InsertLogNote` that allows consistent entry to the DNN Schedule Item History log. Using this as a storage mechanism works well, as over time DNN will automatically clear old results based on the cache duration settings.

Once this has been created and installed, the job will need to be configured as a new scheduled task inside of the Host  Scheduler menu item.

Controlling Navigation and Module Views

Although they've been covered in other chapters at a high level, it is important to discuss a few specifics related to navigation and multiple views in DNN.

Isolation Mode and Module Controls

In [Chapter 13](#)'s example, a link was created using `ModuleContext.EditUrl("MyOtherView")`. It opened a link to another view control as defined in the `.dnn` manifest. This works well and will result in a secondary control being displayed to the users. However, there is an unintended consequence when doing this. Other modules on the page disappear. In many cases, this is acceptable behavior. For an example of when it's acceptable, navigate to the Community Forum at dnnsoftware.com/forums and scroll to the very bottom of the page. You will see the content as outlined in [Figure 15.4](#).

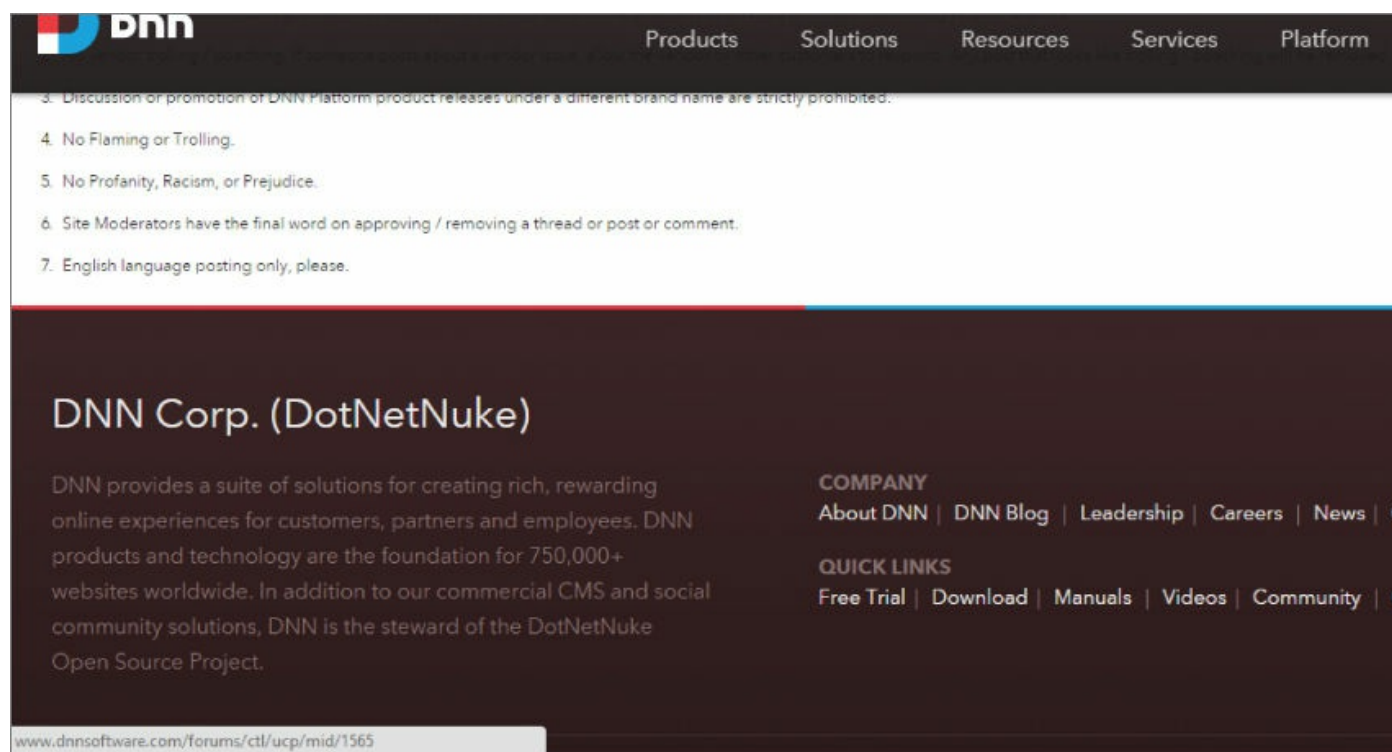
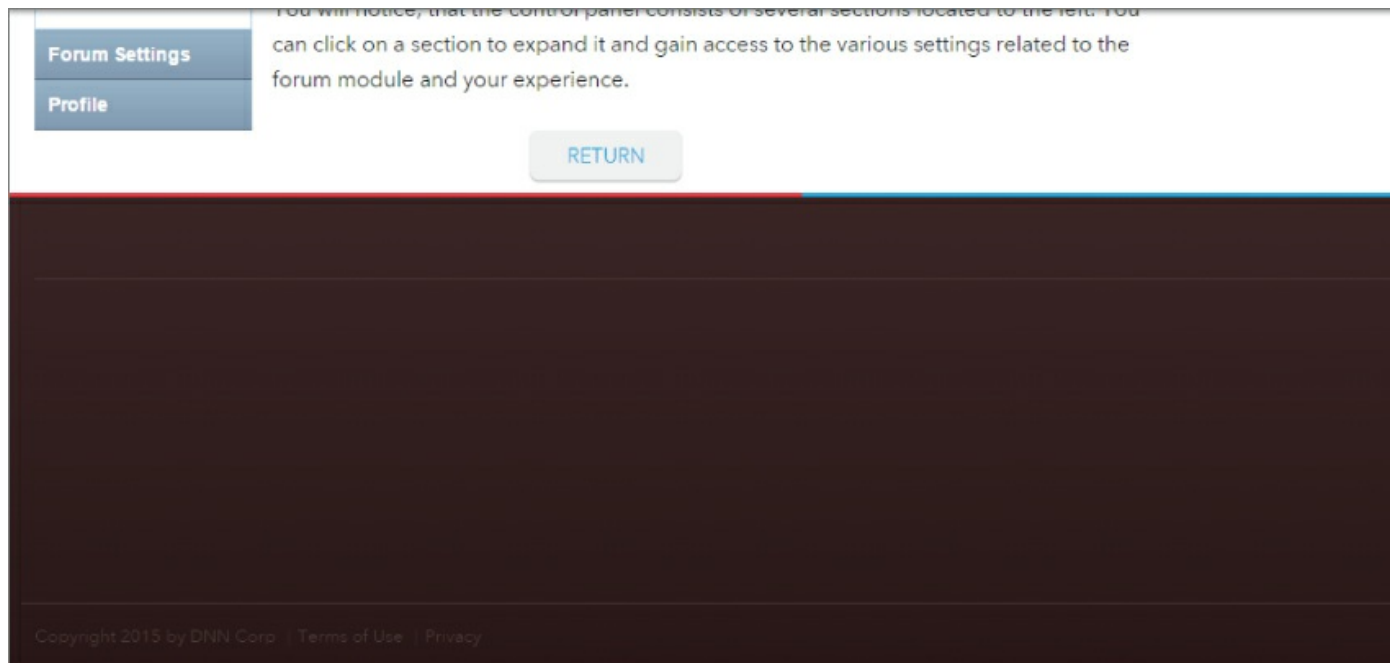


Figure 15.4

At the bottom of the forum content, you will see a HTML module that defines the rules, as well as the footer content with DNN Corporation marketing information and quick links. Scroll back to the top of the page and select My Settings to change the configuration of your forum user. The bottom of the

page will now resemble [Figure 15.5](#).



[Figure 15.5](#)

All the footer content is gone in this view, even the copyright! However, it is easy to add another `.ascx` file and call `EditUrl` to generate a link to it and be done. The important thing to understand is that there are trade-offs.

Bypassing Isolation

Now that you understand what isolation is, you might wonder how you can get around it. This is an aspect of DNN development that does not necessarily have a “best practice” around it. Nor does the framework have any real built-in solution recommendations. This section quickly discusses two commonly used implementations for bypassing isolation.

Dynamically Loading Controls

One of the more commonly favored ways to bypass isolation is to utilize an ASP.NET placeholder control declared in the `.ascx` file. Then within the code-behind in the `pageinit` method, investigate the request and load the appropriate control for the current user's action. This solution works well and does not come with any additional overhead.

The trade-off is that if the events are out of order, viewstate and some other ASP.NET features might not behave as desired.

Toggling Visibility of Sub-Controls

Another commonly favored solution is to have a main view control with multiple sub-controls. Based on the user's selection/state in the environment, you can simply toggle the visibility of the view in question. You can bring something into focus or remove something from view if needed. This route works well; it is fast, simple, and doesn't require much planning or forethought.

However, the trade-off of this solution is that the other controls, if left alone, will still consume viewstate and could greatly increase page size/load time in overly complex view situations.



CAUTION

Module isolation is triggered by the existence of a `mid` and `ctl` parameter in the querystring for the current page. When you're developing a scheme for managing interface views, avoid using any URL values involving `mid` or `ctl`. Failure to do so might result in unexpected DNN behavior.

Managing Linking

When you're linking from page-to-page or control-to-control, there are three options available—`Globals.NavigateURL()`, `PortalModuleBase.EditUrl()`, and `ModuleContext.EditUrl()`. It is important to consider that `NavigateURL` is unaware of the settings for `supportsPopUps` to allow controls to render in a dialog window. Therefore, if you're working with multiple manifest controls, `EditUrl` is the option that you should use.

Additionally, as a quick tip, [Listing 15.13](#) shows how to pass additional querystring information to the current page without triggering isolation mode.

[Listing 15.13](#): NavigateUrl with Extra Content

```
Globals.NavigateURL(TabId, string.Empty, "mode=fragment",  
"name=custom")
```

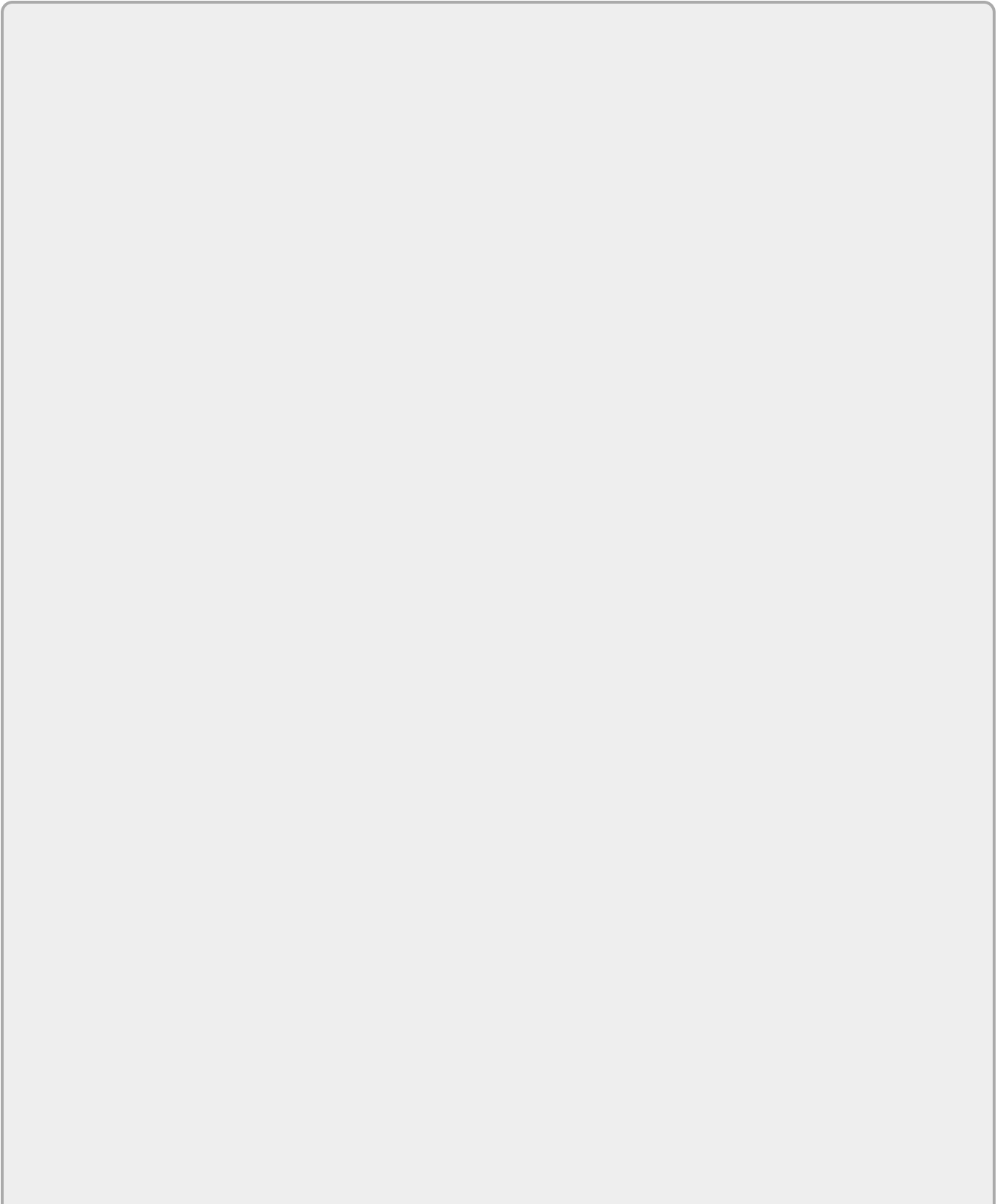
When you call `NavigateURL` in this manner, the querystring values are rewritten into the final URL. Assuming that `TabId` has the value for `/Home`, the generated URL would be `/Home/mode/fragment/name/custom`, which gives you a very nice URL with minimal effort.

Summary

This chapter dove head first into the DNN APIs and explained where the classes are located and the common methods to interact with them. The chapter reviewed advanced module functionality, such as Web API services and scheduled jobs, and finished with a review of navigation and page management.

Chapter 16

Developing Modules: Best Practices and Moving Forward



What You Will Learn in This Chapter

- Managing project compilation and versioning
- Managing project dependencies
- Identifying Azure Support
- Interacting with the database appropriately
- Using the Extension Verification Service
- Preparing for DNN neXt

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 16](#) download and individually named according to the names throughout the chapter.

After spending the last few chapters reviewing different techniques and methods for interacting with DNN, it's time to shift focus to high-level best practices and formats. It is great to be able to deliver an extension that works with DNN; however, it is a whole different story to be able to build something that is reusable and comprehensive and that allows your solution to stand the test of time and work across multiple types of DNN installations. This chapter steps through a number of scenarios and discusses special considerations that you should account for when developing any DNN solution.

Managing DNN References and Versions

Some of the most complex decisions DNN developers must make relate to the process of managing and selecting the various DNN assemblies when building extensions. You have to ensure that the most “proper” assembly version is selected and that the references are easy to manage.

Which Version to Reference

The first question to address is which version of DNN should be referenced as part of your extension(s). At the time of the writing of this book, DNN 7.4.0 is the most current version of the platform. By the time you read this chapter, 7.4.1 or maybe even 7.4.2 will be available. The process of selecting the version to use as part of a project is a multi-faceted one. Note that there is an ever-complex balancing of trade-offs and impacts of one version over another.

Looking into this, it is important to remember what exactly a version number selection does. Once you select a version, you have set the “minimum” version for that particular project. For example, if you compile against DNN 7.4.0, it is safe to assume that your project will work if installed to 7.4.0 as well as any version greater than 7.4.0. This is possible thanks to the long-standing tradition of the DNN Platform to ensure backward compatibility for prior version features and functions. (See the DNN neXt discussion later in this chapter for an important note.)

In the case of a custom-built solution for internal usage only, there is no harm using the then-current version of DNN for any custom modules. However, if you're developing solutions for a broad audience—for example, for a commercial component—you might need a different strategy. At this time, developers often compile a module using DNN 7.0.0 or 7.2.0 for commercial elements, depending on the particular features and functions that are desired as part of the solution.

The key in making the proper decision is to ensure that the needed APIs and functionality exist to allow your solution to accomplish its published goals while supporting the broadest range of installations. It is recommended at minimum to support the initial release of a particular version and beyond. For example, support 7.2.0 and later, not 7.2.1 and later. This helps to avoid confusion with your users.

Enforcing the Dependency

Selecting the version of DNN to compile against is only the first half of the battle; the other portion is ensuring that users can't install your extension on an environment that doesn't support it. [Listing 16.1](#) shows a snippet of XML that should be loaded inside of the `<package>` node of your `.dnn` manifest.

[Listing 16.1](#): Core Version Dependency Enforcement

```
<dependencies>
  <dependency type="CoreVersion">07.00.00</dependency>
</dependencies>
```

By making this small addition to your manifest, you prevent the unfortunate issue of installing a module that requires a version of the core that does not exist. This is critical, as installation to a prior version will typically result in the site not loading for the user, until your `.dll` file is manually removed.

Making Things Easier with NuGet

Depending on the particular module development template you use, it is important to note that there are numerous DNN NuGet packages that are maintained on a regular basis. Using these packages makes it is easy to manage the references to DNN in the same manner that ASP.NET and related projects manage their references. If you're looking to build “outside” of the structure or want true portability of your developed solutions, these packages are invaluable. Distributed through NuGet.org, there are three supported packages that are distributed by DNN:

- **DotNetNuke.Core:** All DNN extension projects should reference this package. It contains `DotNetNuke.dll` as well as the reference to `Microsoft.ApplicationBlocks.Data.dll` that's needed for DAL and DAL+ implementations of data access.
- **DotNetNuke.Web:** This package references additional utilities for working with web components. Items such as Telerik and `DotNetNuke.WebControls.dll` are included in this optional package.
- **DotNetNuke.WebApi:** This package adds all needed references to properly construct and work with the DNN implementation of the Web API in a custom extension product.

There are a few important elements to take into consideration as they relate

to these NuGet packages. First, the package names reflect the older DotNetNuke name and not the newer DNN moniker. Given the upgradability of NuGet packages, this behavior will most likely not change in the near future. This will ensure that existing individuals can upgrade seamlessly to new versions of the packages.

Second, prior versions of applicable packages are available, which allows developers to benefit from using NuGet packages while still targeting older versions of DNN. Reviewing the package history on [NuGet.org](https://www.nuget.org) shows that most packages have support for 6.0.0, 7.0.0, and major versions forward.

Once you identify your desired package, simply use the NuGet Package Manager console in Visual Studio (choose Tools ➤ NuGet Package Manager ➤ Package Manager Console) to install. For example, to add DotNetNuke.Core version 6.0.0 to your project, you would enter the following command:

```
Install-Package DotNetNuke.Core -Version 6.0.0
```

Managing External Dependencies

With DNN references properly managed and core versions properly limited, it is important to look at managing other dependencies. In a dynamic environment such as DNN, individuals will often want to install a module without necessarily reading all of the documentation. Therefore, it is important to ensure that developed extensions declare all dependencies, or prerequisites, properly to prevent accidental installation. The dependency validation process, which is part of DNN's module packager, provides great freedom over the specification and validation of any needed dependencies. [Listing 16.2](#) shows a few additional dependency options that you can use as part of an extension's DNN manifest. These can be used in conjunction with other dependencies, as needed.

[Listing 16.2](#): Additional Dependency Examples

```
<dependencies>
  <dependency type="package">Wrox.Guestbook</dependency>
  <dependency type="managedPackage"
    version="1.2.0">Wrox.Suggestions</dependency>
  <dependency
type="type">System.Reflection.ReflectionContext</dependency>
</dependencies>
```

You can see from this example that three new dependency types have been introduced. A `package` dependency verifies that a DNN package with the specified name is installed. The version number of the installed package has no bearing on the successful validation of this dependency. A `managedPackage` dependency type ensures that a specific minimum version of a package is installed. From [Listing 16.2](#), you can see that the defined `managedPackage` dependency requires version 01.02.00 or later of the `Wrox.Suggestions` module to be installed. Leading zeroes in module version numbers are not needed as part of the dependency definition.

The final type of dependency is a bit more complex. This is a *type* dependency. This dependency has multiple purposes, and the primary consideration here is that DNN must be able to locate the defined type as part of the running application content. The example in [Listing 16.2](#) uses the `ReflectionContext` class as the example. It validates that the DNN installation is running on a server that is using .NET 4.5 or later, as the `ReflectionContext` class was

added as part of .NET 4.5. `System.Tuple` could be used to validate that the installation is running on .NET 4.0, and any other type could be used to validate items such as deployed third-party DLLs or related functionality.

Specifying proper dependencies for extensions at installation time is a critical need to ensure reliability of the destination installation. When there is a failure during installation, the user is presented with a detailed message outlining the nature of the error.

Future-Proofing Data Interactions

Now that you have learned how to prevent site administrators from installing your extensions when they might not work, it is important to consider other development practices that you can use to ensure a smooth development process. The DNN API is expansive, and for those new to the platform it is often hard to fully comprehend the best places to start looking for information. This book has multiple chapters that are dedicated to development within DNN and that discuss various API interactions.

Bypassing the API

One of the most common questions that appear in online forums as it relates to development with DNN is, “Can't I just go to the database for what I want?” This question has various answers. Given that this chapter is focused on best practices and future-proofing, the answer is quite simple. Theoretically, yes, it may be easier to create a database query that quickly pulls a list of roles to use in a custom module. However, that “speed of development” comes at a pretty steep cost.

By directly querying and interacting with DNN database objects, your developed solution is no longer dependent on an API, but it is hard-coded and entrenched into the data model. Any modification to the data model, an action *not guaranteed* to be future proof, not only places your application at risk but additionally could place the users' installation at risk for upgrades. As such, it is recommended that APIs be used in lieu of any direct database interactions. This has been important for a number of years as the platform has grown and is even more important with DNN neXt on the horizon!

Managing Foreign Keys

When it comes to proper data management, there is nothing worse than a module that adds data to the database and doesn't account for any proper cleanup. [Chapters 13](#) and [14](#) explained how data can be tied to a `ModuleId`, `TabId`, or `UserId` depending on the desired segmentation of data. An often-overlooked portion of this data association is how that data interaction is completed. Consider the two `CREATE TABLE` statements in [Listing 16.3](#).

[Listing 16.3](#): CREATE TABLE Examples

```

CREATE TABLE {databaseOwner}[{objectQualifier}GoodTable] (
    ItemId INT IDENTITY(1,1) PRIMARY KEY,
    ModuleId INT CONSTRAINT [FK_GoodTable] FOREIGN KEY REFERENCES
        {databaseOwner}[{objectQualifier}Modules] (ModuleId) ON
DELETE CASCADE,
    MyData VARCHAR(500)
)

CREATE TABLE {databaseOwner}[{objectQualifier}BadTable] (
    ItemId INT IDENTITY(1,1) PRIMARY KEY,
    ModuleId INT CONSTRAINT [FK_BadTable] FOREIGN KEY REFERENCES
        {databaseOwner}[{objectQualifier}Modules] (ModuleId),
    MyData VARCHAR(500)
)

```

With these two tables, you can see one slight difference. They both create a foreign key relationship between the newly created table and the DNN modules table. However, only one of these adds the `ON DELETE CASCADE`.

When interacting with DNN and working with foreign keys, it is important to ensure that if your interaction works with DNN data and adds references to it and if DNN needs to remove that element from the database, your code must not only *not* stop this from happening but should additionally clean up your data. Specifically reviewing the behavior of a user deleting a module from a page, the initial action only changes the `IsDeleted` flag to 1 and the module now appears in the recycle bin. If the users change their minds, they can easily restore the module and all of the data will still be there.

If a user decided to empty the recycle bin, the `BadTable` in [Listing 16.3](#) would introduce problems. DNN would attempt to remove the module record for the deleted item and would encounter a foreign key issue. DNN would be unable to clear the item from the recycle bin. Manual intervention at the database level would be required at this point by the user. The user would have to intervene before operations would be allowed to continue. This situation could be further exacerbated if the user needed to delete an entire site, which required removing hundreds of pages and hundreds of associated modules.

Supporting `objectQualifier` and `databaseOwner`

In addition to properly using APIs, there is another aspect of DNN interactions that is still a primary concern. This is supporting the `{objectQualifier}` and `{databaseOwner}` replacement tokens in all generated SQL scripts. If you look on the Internet, you'll see that there are many schools

of thought related to this functionality. The true “must-use” scenarios are most likely very small; however, where installations are using these configuration elements, it is imperative that all generated scripts support them.

As such, this will limit the data access strategies that can be used as part of an installation. DNN's DAL, DAL+, and DAL2 methodologies all support these tokens. However, popular ORMs such as EntityFramework or NHibernate do not.

Let's quickly revisit the purpose of these elements, which were first discussed in [Chapter 13](#). The `{databaseOwner}` replacement token defaults to a value of `dbo` and is used as the schema of any created object. The `{objectQualifier}` is a prefix to the actual object name. The current default for this replacement token is a null value, resulting in no change. However, older installations may use `dnn_`, as this was a common recommendation. To properly support these replacement tokens, you must structure any references to database objects in the `SqlDataProvider` files in the following fashion:

```
{databaseOwner} [{objectQualifier}OBJECTNAME]
```

where `OBJECTNAME` is the name of the target object. It should be noted that the brackets, `[]`, included here are *imperative* for proper operations.

If you elect not to support this development paradigm, it's very important that you properly communicate this limitation to any potential users.

Supporting Azure

Using Azure, SQL Azure, and Azure Websites is something that is becoming more and more common. DNN has fully supported running on Azure and specifically using SQL Azure for the database backend for a while. If your extension can work in this environment, it is important to add the code snippet in [Listing 16.4](#) to your module's manifest.

[Listing 16.4](#): Noting Azure Support

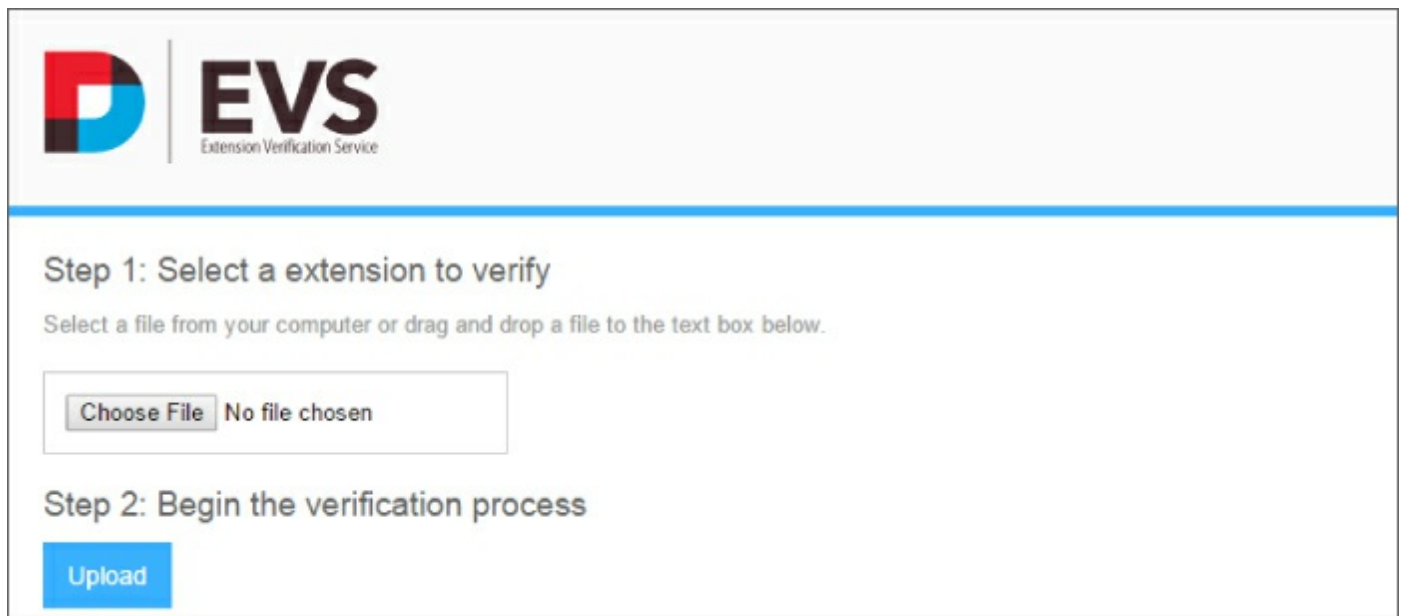
```
<azureCompatible>true</azureCompatible>
```

If this tag is omitted and users attempt to install the extension in a Windows Azure environment, they will be notified that the module may not be

compatible with SQL Azure and will be forced to confirm the installation. If you are unsure of your module's support for SQL Azure, use the Extension Verification Service (EVS) to aid in the identification.

Extension Verification Service (EVS)

To help developers evaluate their solutions against a set of “best practices,” DNN Software created and maintains the Extension Verification Service. Hosted at <http://evs.dnnsoftware.com>, this online service allows developers to validate their extension against a number of rules. At a high level, this validation service takes an uploaded module, installs it, and then uninstalls it. It scans for any errors during installation and any missing manifest information and ensures a clean un-installation process. [Figure 16.1](#) shows the welcome page of the EVS site.



[Figure 16.1](#)

The process is quite simple. You click the Choose File button to locate your installation ZIP file and select Upload. After opting to upload your extension, you wait. It will take a while for the results to come back, depending on the number of other requests, the size of the module, and other environmental factors. *Do not* navigate away from the EVS website during this process. If you do this, you will not be allowed to see any of the results from the verification and would need to start the process over again.

What It Validates

The EVS system checks for a number of elements and validates the successful installation and integration of a module at a high level on multiple versions of DNN, including 6.0.0, 7.0.0, 7.1.0, 7.2.0, and 7.3.0, if they're supported as part

of the extension being tested. In addition to validating the successful installation or un-installation of the extension, EVS validates that all needed metadata is properly accounted for, resulting in a complete and valid package.

EVS additionally validates support of SQL Azure first by scanning any included `SqlDataProvider` file for proper SQL Azure syntax and additionally by executing a systematic Azure Backup and Restore. This step is critical as it is possible for an extension to be installed; however, after installation, it could impact the user's ability to properly back up/restore the site.

Interpreting the Results

When interpreting the results from EVS, it is important that you remember that it is an automated system and involves simply checking boxes and indicating “yes” or “no” to a set of conditions. When you're validating custom extensions against the service, there are often issues that an automated system can't consider, such as usage and deployment. [Figure 16.2](#) shows the high-level results screen of a random third-party extension that ran through the EVS system.

The screenshot shows the EVS results interface. On the left, a table lists package details:

Package Name	ICG Modules IpBasedAutoL...
Package Type	Module
Package Version	01.00.00
Min DotNetNuke Version	06.00.02
Min .Net Version	3.5
Package Has Errors	True

On the right, there are options to download results in CSV, XLS, or XML formats, an email subscription form with a 'submit' button, social sharing icons for Facebook, LinkedIn, and Twitter, a 'Go to Forum' button, and a 'Download' link for the 'Azure Checker Output'.

Below the main content, a summary of messages is shown:

- Errors (1)
- Warnings (14)
- Info Messages (12)
- System Errors (0)

[Figure 16.2](#)

This display shows a few key elements worth noting. First, notice that the

high-level information regarding the installation page is restated at the top of the page. This ensures that the high-level information is evident. Download options are great for sharing results with others, as is the Azure Checker Output option. Let's step through the various sections of the results screen and discuss the types of issues that might arise.

Errors

When you're validating extensions, it is imperative to validate any reported error messages. Items tagged in this section are critical elements that EVS believes could prevent the extension from properly installing. Each error contains a message and a rule. For example, if your manifest references a file that could not be located in the installation, the rule

`PackageVerification.Rules.TwoWayFileChecker` is triggered, as EVS was not able to validate files.

Warnings

Warnings are constructed to ensure that developers are aware of situations that could become problematic if they aren't properly managed. [Figure 16.3](#) shows one example of a warning message.



[Figure 16.3](#)

In this case, the notification outlines that a database script as part of the module adds a dependency to the core `Modules` table. Upon further review, you can see that `ON DELETE CASCADE` is properly used, which mitigates this issue. The warning in this situation is there so that the developer will verify that this situation is intentional.

However, other elements that can occur because warnings include situations where a SQL script omitted an `{objectQualifier}` or other related token value. If a module is compatible with Azure and the manifest does not declare this support, it will also appear as an error.

Information Messages

With successful verification of execution, a number of messages will be displayed in this section. These messages include informational data, such as the time needed to execute the Azure backup, the time needed to execute the Azure restore, and confirmation of key manifest information, such as author and manifest structure.

System Errors

All tests should execute without any recorded system errors. Treat a system error as a complete failure of the validation test. Additional validation should be completed against the manifest package.

Getting Prepared for DNN neXt

With the recent release of DNN 7.4.0, there is a lot of community buzz around the future of DNN. There is talk of major changes in 7.5.0, with first-class support of MVC being billed as one of the key deliverables for that release. With DNN neXt staring down at us after the 7.5.0 release, what does this mean for developers?

Nothing is known for sure at this time; however, a few general rules of thumb can be applied to current development. These are basic rules that, regardless of the future pathway, ensure that developed solutions have the best chance of migration.

Avoiding Deprecated Methods

The DNN API has been constantly evolving for more than 12 years now and, as you can imagine, there has been a plethora of change over these years. Up to the current 7.4.0 release, there has been strong support for backward compatibility. As part of this support, methods were marked as deprecated years ago, even though their internal implementations were updated to use the new methodology. The current plan is that deprecated methods will not be included in DNN neXt. Additionally, DNN 7.5 will mark many more methods as deprecated that are not expected to make the transition.

It is best practice to assume that a method, class, or other deprecated object could be removed in future releases. As such, it is strongly recommended that you verify your solution against the latest releases of DNN and work to remove any calls to deprecated methods. Your developed solution will more likely be future proof. The new methods are often more efficient than the older, deprecated options.

Validation of any deprecated method usage is simple. Simply build your project and review the error list. Any deprecated method calls will be included as warnings.

Following Best Practices

All the practices included in this chapter as well as the items validated as part of the EVS provide guidance to ensure that your developed solutions are as close to “on target” as possible for the future direction of DNN. The DNN Wiki is another source of help, as it explains best practices and procedures. By

using methods that are common to the larger DNN community, you ensure that a simpler pathway to migration is developed that you can leverage when you move to DNN neXt.

Staying Active

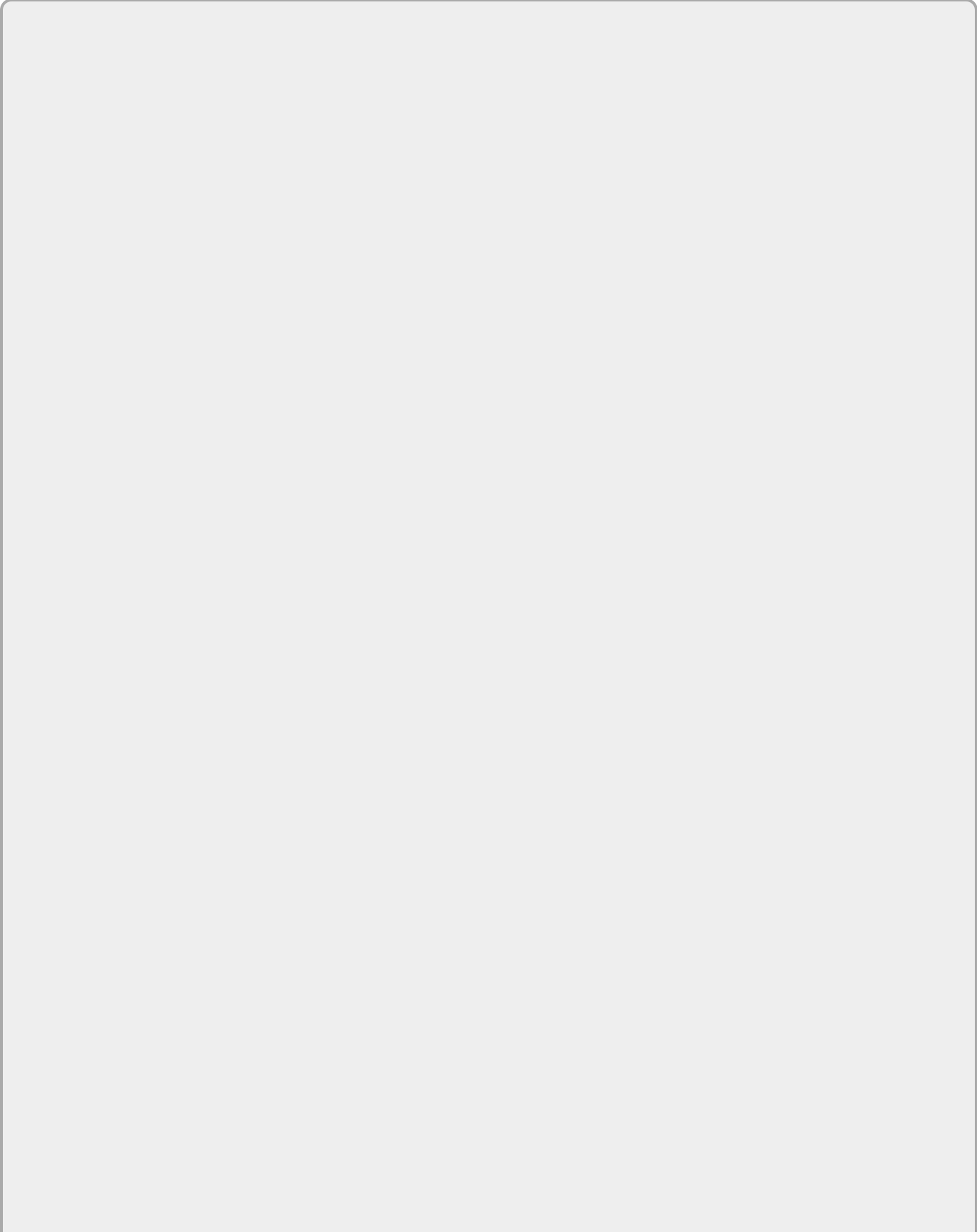
DNN has undergone revitalization in the open source community, and more steam is being gained on this front. As such, the discussions about what is next are being held in open communication channels. Getting involved in discussions and keeping up on the happenings on DNNSoftware.com under the Community Voice, Community Forums, and Community Blogs sections will allow you to take a front-row seat on the adventure that will be the next releases of DNN.

In addition to your commentary and direction decisions, you can also jump on GitHub and submit enhancement requests to the platform. By staying active with the product, you will be able to adjust development methodology and processes so that they more closely align with any future releases.

Summary

This chapter explored a number of recommendations to help ensure that your developed solutions on the DNN platform are created so that they work well, not only now but in the future as well. You explored tools that you can use to validate the structure and suitability of deployment of extensions. Additionally, you learned about dependencies and how they can be used to prevent issues from arising should an expected component be missing.

Chapter 17
Skinning



What You Will Learn in This Chapter

- Creating a DNN skin based on modern web practices
- Creating a responsive, mobile-friendly skin
- Setting up a skinning environment with Visual Studio
- Packaging a skin for installation on other DNN sites

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wiley.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 17](#) download and individually named according to the names throughout the chapter.

What is skinning in DNN? Skinning is creating a consistent and reusable template for the layout and design of your DNN site. Skinning allows designers and front-end developers with no knowledge of server-side development to create dynamic HTML and CSS templates. This allows for a separation of concerns for development and design. Experts in HTML, CSS, and JS can create interesting and amazing designs that content editors can use to add new pages and content while keeping the look and feel consistent throughout the website. This chapter will take you through the entire process of developing a skin by leveraging what you already know about creating web pages and explain the concepts needed to plan and convert your own designs into DNN skins.

Skinning by Today's Standards

One thing we all know is that the web is constantly changing. HTML5 has now become an official WC3 Recommendation, and as of DNN 7, you are now free to use those HTML5 elements to define content sections and repeatable content blocks following today's best practices. While you are free (and encouraged) to use HTML5 to create your DNN skin, it is important to realize that DNN 7 supports all major browsers, including Internet Explorer version 8 and up. This means that DNN still supports a browser that does not recognize HTML5 elements. We will discuss how to resolve this issue once you begin creating your skin.

Parts of a DNN Skin

What is often referred to as a DNN skin is really a collection of the parts that make up the design of your site. There are several items related to skins that you need to understand before you can begin creating one.

Skins

A *skin* is a file with HTML and some additional components that allow for adding dynamic content that defines the layout and styling of one or more pages. A skin should be able to stand on its own when moved from one site or DNN installation to another. It should not have any dependencies on a specific site. However, it can depend on third-party skin objects that can be installed along with an installable skin package (both of which are discussed in the sections that follow).

Containers

A *container* is a file with HTML and some additional components that allow for displaying dynamic content that defines the styling of a module. A container is typically meant to support a specific skin package, allowing for flexible, varying content styles within a page. A container is similar to a skin, but only styles the content within the module to which it is applied.

Skin Objects

A *skin object* is a user control that can be used to provide dynamic content on the skin. A skin object will be replaced by the content that it is created to provide at runtime. A good example of this is the logo skin object. A logo skin object can be placed on a skin and will be replaced by the logo that is applied to the site by the site's administrator in the Site Settings. Skin objects are similar to modules, differing in the fact that they do not have any settings through DNN's user interface, simplifying their use over a module and allowing them to be defined within a skin. However, each skin object does have its own parameters that can be defined within the skin. Some examples of skin objects include breadcrumb, copyright, language, login, logo, search, and styles.

Skin Packages

A *skin package* is a collection of all the individual skin and container files, as well as the supporting stylesheets, fonts, images, JavaScript, and any other resource required for the skin to be used. Skin packages typically include three or more skins (home page, internal page, and admin pages) that have similar styles (colors, fonts, and so on) but provide different layout options. These skin packages cannot be installed through the DNN interface but must be added manually to the site either through FTP or direct access to the DNN installation's folder structure.

Installable Skin Packages

An *installable skin package* is an extension that's a ZIP file containing a skin package as defined previously. These are installed through the DNN user interface and must be installed by a host user.

Content Panes

Content panes are designated locations within a DNN skin where modules can be placed onto a page in a DNN site as shown in [Figure 17.1](#). A content pane is used to place content on a page that needs to be edited by a content editor, through the DNN interface. Multiple modules and module types can be placed on a page within a content pane. DNN requires that there be at least one content pane on a skin, and it must be named `ContentPane` (not case-sensitive). Aside from the default content pane, as many as necessary can be added to a skin.

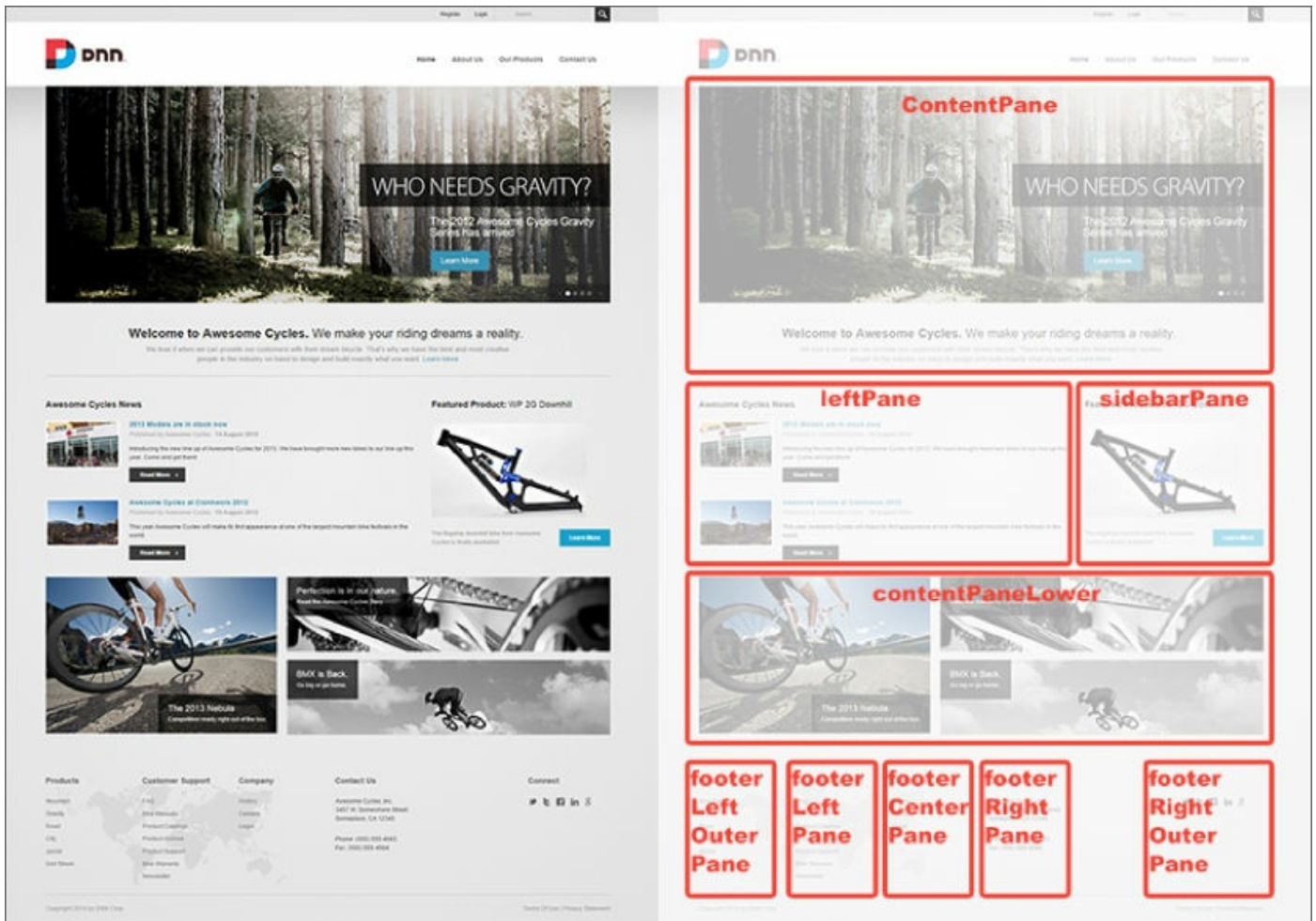


Figure 17.1

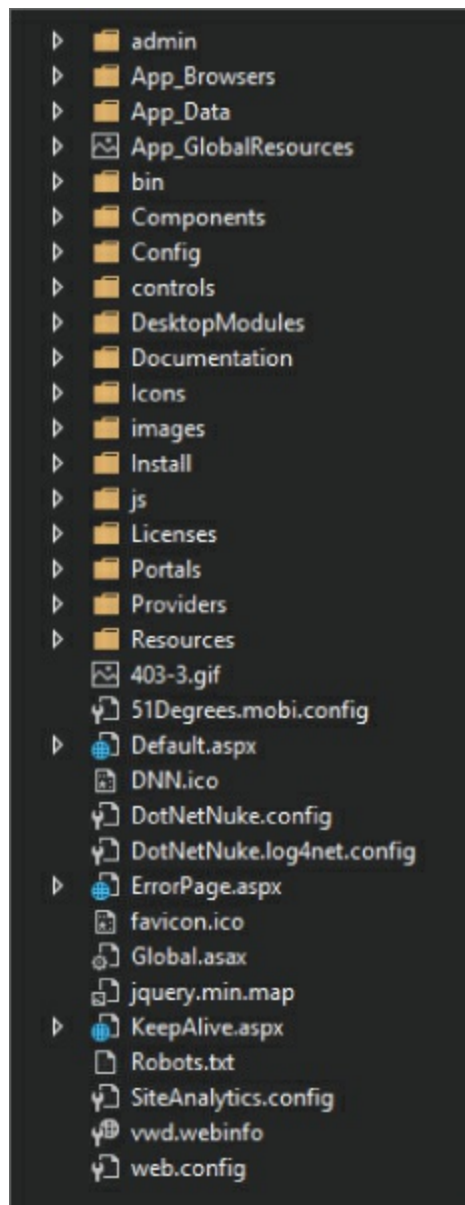
Navigation

The navigation model in DNN provides the ability to create a dynamic navigation that will be updated automatically as pages are added to the site. This navigation can also hide and disable links to pages through the DNN user interface. The DNN navigation model has a dynamic navigation that can support several navigation providers. The primary default navigation provider that is packaged with DNN is the DDR Menu. This menu allows for a site's navigation to be rendered to the page based on a customizable template, thereby giving great control to the layout and functionality of the menu.

Folder Structures

Organization of files within DNN is very important. DNN requires that files be placed in certain locations to perform their intended functions. DNN monitors those folders and folder structures to provide simple integration

with skin and container packages. See [Figure 17.2](#) for an example of the root folder structure in DNN.



[Figure 17.2](#)

Skin Locations and Permissions

DNN requires that skin packages be placed in a specific location to be used by the skinning engine. The skinning engine then applies the assigned skin to each page at runtime. Skins packages can be made to be available to either one specific site or to all sites in a DNN installation. This is done by placing the files for the skins in specific locations within the folder structure of the installation.

To understand how a skin's access is determined, you must first understand some basic folder structure for your DNN installation. All site-related files are located in the `Portals` folder in the root of the installation (see [Figure 17.3](#)). DNN uses the name `Portals` in the API to relate to a site. Each site you create will have a corresponding folder inside the `Portals` folder. These are typically named for the `portalId` assigned to each site when they are created. This will be the name of the folder associated with the site. By default, the first site is assigned an ID of 0 and a corresponding folder name of 0 in the `Portals` folder. The next site will have the `portalId` of 1, a corresponding folder name of 1, and so on. All related skins, containers, images, and so on, will be located in this folder. DNN monitors the site folder's contents for the `Skins` and `Containers` folders. Skins and containers placed in these folders will be made available only to the corresponding site for the site's folder in which it is located. For example, if a skin package is placed in `~/Portals/1/Skins` folder, all skins in that skin package will only be made available to the site with a `portalId` of 1. Any other site, such as the first site with the `portalId` of 0, will not be able to apply that skin to any of its pages.

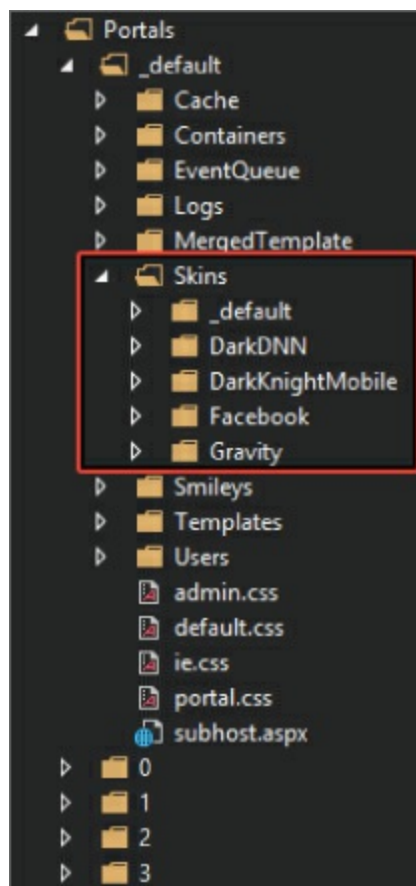


Figure 17.3

If you want to make a skin package available to all sites within the DNN installation, that skin folder can be placed in the `~/Portals/_default/Skins` folder. This would be considered installed at the “host level,” meaning that it reaches across all sites, as does a host's security access, as shown in [Figure 17.4](#).

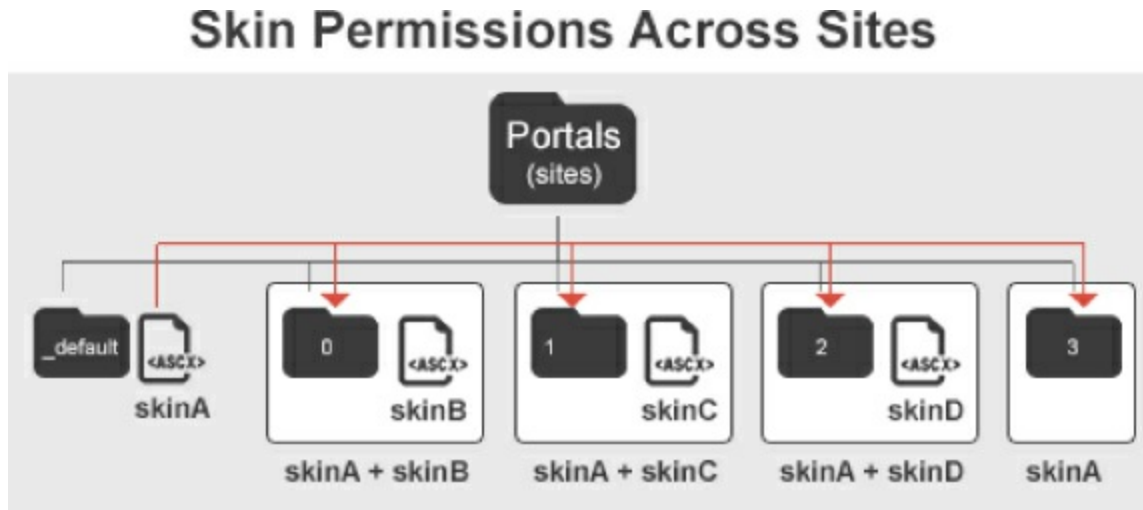


Figure 17.4

Container Locations

Containers are subject to the same permission rules as skins (see [Figure 17.5](#)). They are available only to the sites associated to the folders in which they are located. So, a container package specifically for a site with a `portalId` of 1 would be located in the `~/Portals/1/Containers` folder.

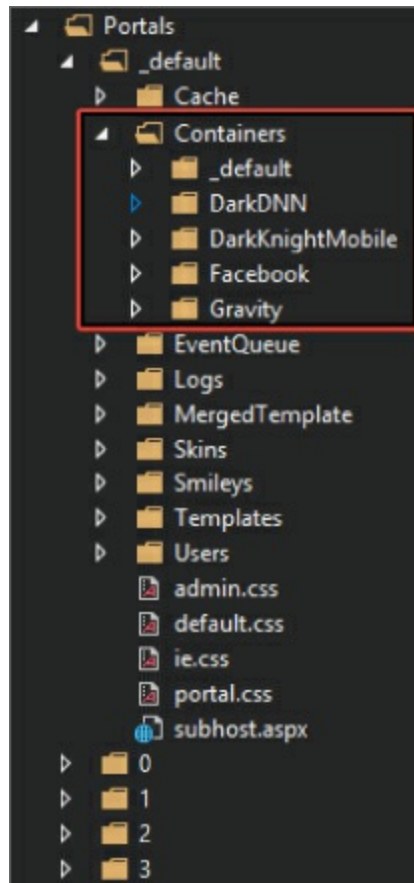


Figure 17.5

Skinning Approaches

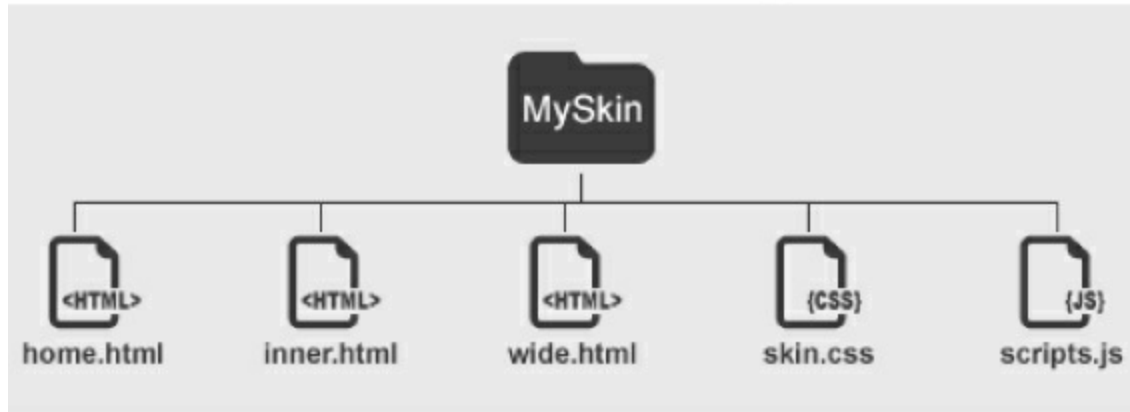
When creating skins, there are two techniques available to the skin developer. The first is the *HTML skinning method*, which can be easier to a beginner but requires more upkeep and steps to make accessible for use on DNN pages. The second approach is the *ASCX skinning method*. This technique is slightly more advanced but is readily available to view within a DNN website. We will take a brief look at how each method is developed and the pros and cons of each. This book takes the ASCX approach for examples moving forward, as it's much more efficient.

HTML

The HTML approach uses an HTML file that is parsed by DNN to convert it to a skin that can be used by DNN on your site. The HTML file uses objects with parameter elements to define the skin objects for a skin. The parsing engine then creates an ASCX file with user controls (skin objects). Using the HTML approach may be easier at the beginning, but you will more than likely find yourself getting bogged down by needing to continually parse it to convert it to the ASCX format used by the skinning engine. [Figure 17.6](#) shows how the skinning engine parses the HTML files. An example of how to define a skin object with the HTML approach follows.

```
<object id="dnnCOPYRIGHT" codetype="dotnetnuke/server"
codebase="COPYRIGHT">
  <param name="CssClass" value="footer-text" />
</object>
```

HTML Skin Package



Skinning Engine parses HTML files

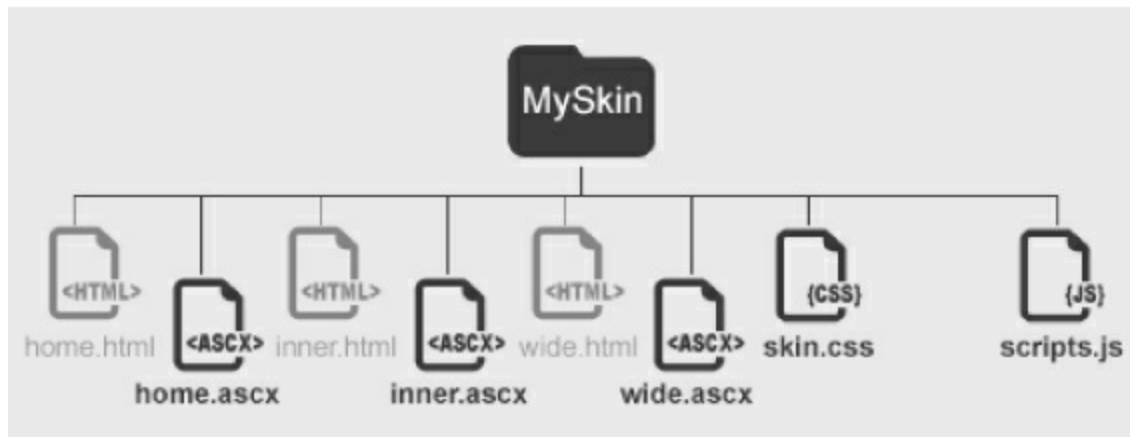


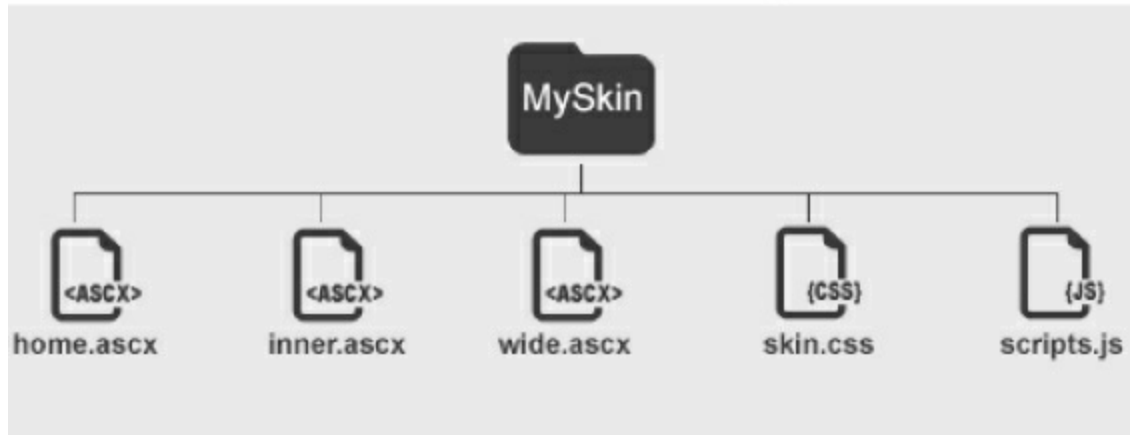
Figure 17.6

ASCX

The ASCX approach uses an ASCX file that does not need to be parsed by DNN as this is exactly what is produced by the parsing engine when taking the HTML approach. This approach bypasses the parsing process and makes viewing updates to a skin much faster. The benefit of using this method is that once the file is saved in the site's skin folder, it is immediately available on the pages where the skin has been applied. The parameters that are specified in the HTML method are done so with attributes for the ASCX skinning method. [Figure 17.7](#) shows the ASCX packaging. An example of how to define a skin object with the ASCX approach follows:

```
<dnn:COPYRIGHT ID="dnnCopyright" runat="server" CssClass="footer-text" />
```


ASCX Skin Package



[Figure 17.7](#)

Preparing to Create a Skin

The best way to learn how to skin is to create one. We will be creating the skin called `SubtleTrend` (see [Figure 17.8](#)). This is a pure CSS skin. No images are needed to create this skin, other than a logo. There will be quite a bit of CSS3 work involved to get the results we are looking for. As mentioned previously, DNN 7 supports modern browsers, including IE8 and up. Because we are providing support for IE8, which does not support CSS3 and HTML5, we will provide some workarounds for the HTML5 and graceful degradation where the CSS3 styling fails in IE8. This will also be a responsive skin, so there will be some size-specific CSS; however, please note that the specifics of the approach to responsive design are outside the scope of this book.



Figure 17.8

Preparing your Skinning Environment

The best way to develop a skin is to create a development environment on the computer you will be working on. Installation is not difficult, and once it's set up, it can be reused to make new skins. Create a local installation of DNN following the instructions in [Chapter 2](#).

Choosing an IDE

DNN skins can be created within the IDE (Integrated Development

Environment) of your choice, such as Microsoft Visual Studio, Adobe Dreamweaver, Sublime Text, Notepad++, or even simply Notepad. For this example, we are going to be using Microsoft's free version of Visual Studio, called Visual Studio Express for the Web (2013 at the time of this writing), as it will provide you with some helpful IntelliSense when creating your skins. To download it, go to <http://www.visualstudio.com/en-us/products/visual-studio-community-vs>. Download and install the latest version for the web.

Setting Up Your Website in Visual Studio

Once you have set up a local installation of DNN and installed Visual Studio, open Visual Studio and select File > Open Website. In the Open Web Site dialog, choose File System on the left navigation. Then, in the file tree to the right, navigate to the file location of your local DNN installation, as shown in [Figure 17.9](#). Using Visual Studio in this manner will provide you with some very helpful IntelliSense when using the ASCX skinning method.

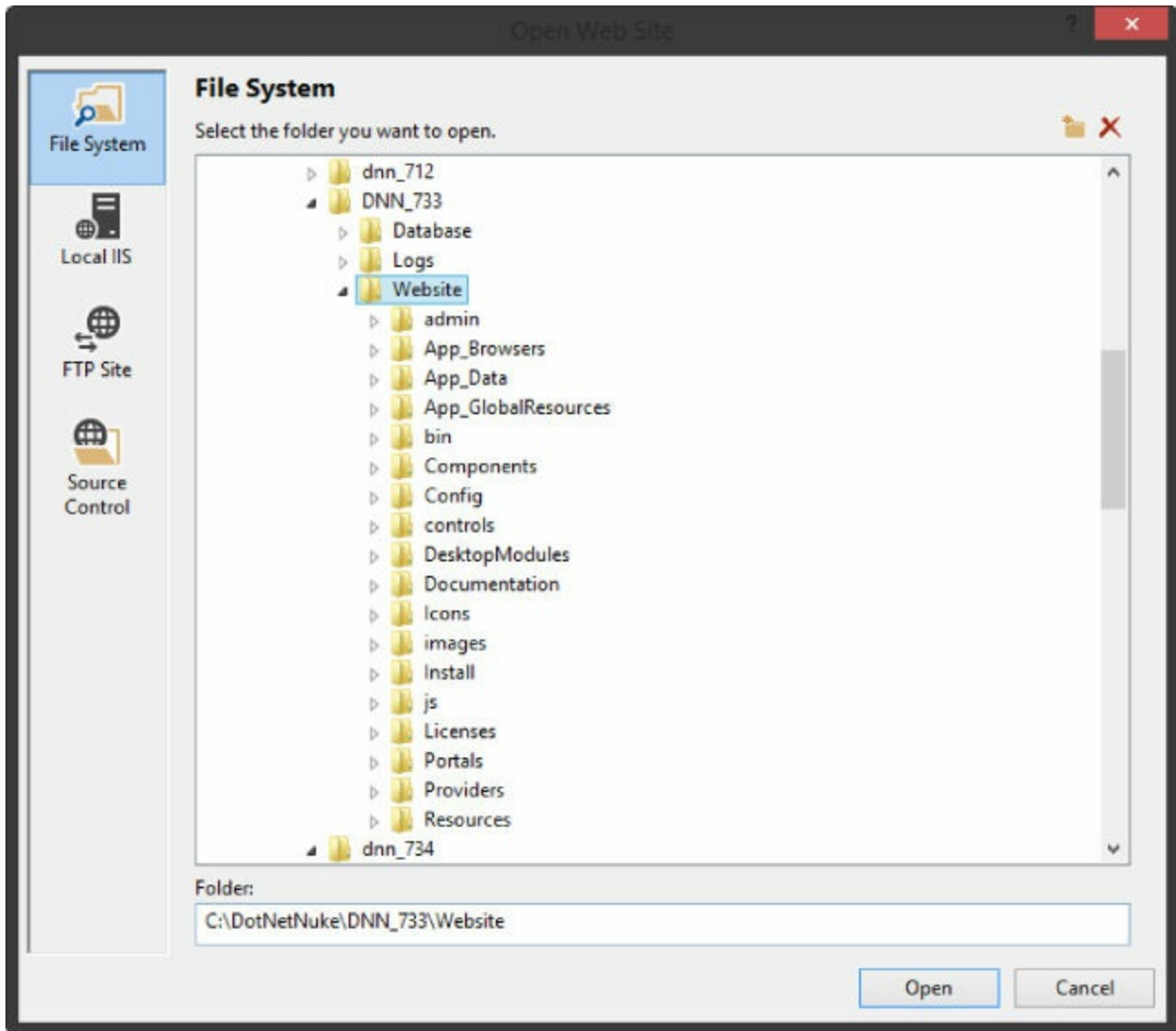
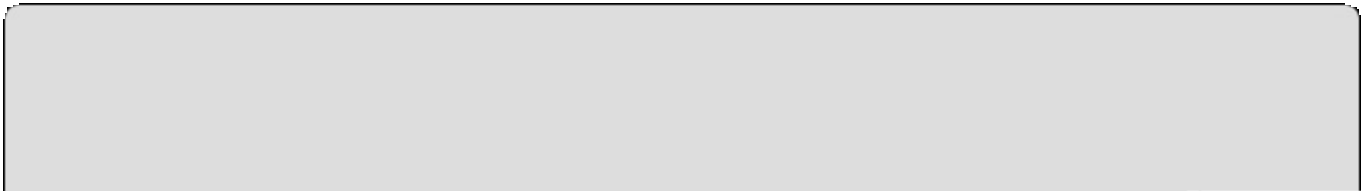


Figure 17.9

Creating the Folder Structure

The first step to creating the skin is deciding which sites will have access to it. You will be creating yours to be available for the entire DNN installation. You will need to create it here to be able to create an installable skin package later in the chapter. Once your website has been opened in Visual Studio, in the Solution Explorer window, expand the Portals folder. (You may have to first turn on the Solution Explorer by navigating to View Other Windows Solution Explorer from the main toolbar in Visual Studio.) There should be two folders in the Portals folder, 0 and `_default`. The 0 folder is the root folder of the first site created with DNN, and the `_default` folder holds resources that are available to all sites created in that specific DNN installation. Expand the `_default` folder, and you will see several folders with

Containers and Skins among them. These folders contain the skins and containers that come preinstalled with DNN. DNN looks for the Skins and Containers folders and will automatically make available any ASCX skins (or parsed HTML skins) in them to the Admin interface, where they can be applied to any DNN page. Next, right-click the new Skins folder and select Add ⇨ New Folder. Name it the same name as your new skin package. This example uses `SubtleTrend`, as shown in [Figure 17.10](#).



NOTE

DNN comes preinstalled with several skins, with Gravity being the default skin for a new DNN installation. This is a good skin to reference when creating your own skins.

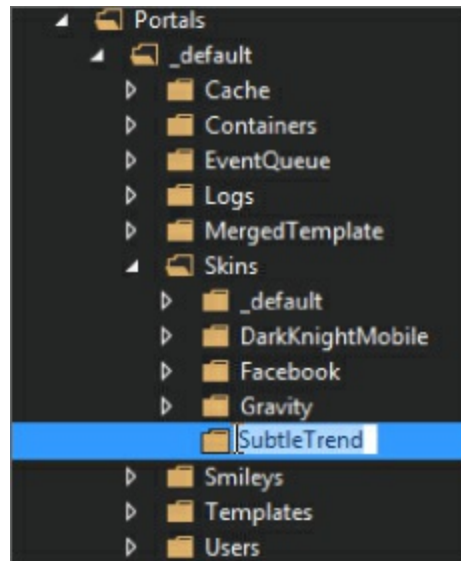


Figure 17.10

Creating Your First Skin

Now that you have everything set up, you can start creating your first skin. You will start with the two approaches for skinning—HTML and ASCX. You will begin with getting the basics of the HTML approach, and then you'll move to the ASCX approach for the buildout.

HTML Approach

Right-click the `SubtleTrend` folder and select Add ➤ Add New Item. Select HTML Page. Give it a name of `mySkin.html` and click the Add button. In the body of the document, add the following code and save the file:

```
...  
<body>  
  <div id="ContentPane" runat="server"></div>  
</body>  
...
```

To make your skin available to DNN, you must first parse it, which will convert it to a DNN skin. To parse an HTML skin, once the skin has been created, you must log in as a host user and navigate to the Admin ➤ Skins page. Find your skin, `SubtleTrend`, in the drop-down. Once selected, scroll down and click the Parse Skin Package button. This will convert the HTML document to an ASCX file. This ASCX file is the same file that you will be creating and editing with the ASCX approach. Anytime you make a change to the HTML document, you must parse your skin by once again navigating to the Admin ➤ Skins page, selecting your skin package, and clicking the Parse Skin Package button to see the effects of your changes. This can become a tedious process, which is why the ASCX approach is much more efficient. You will switch to using the ASCX approach for the rest of this example.

ASCX Approach

You will now take the approach of creating the skin using an ASCX file. Right-click the `SubtleTrend` folder and select Add ➤ Add New Item. Select Web User Control from the C# section and uncheck the Place Code in Separate File option. Give the file the name `Home.ascx` and click the Add button (see [Figure 17.11](#)). Replace all of the code in the file with the following code:

```
<%@ Control language="C#" AutoEventWireup="false  
  Inherits="DotNetNuke.UI.Skins.Skin" %>
```



```
<div id="contentpane" runat="server" />
```

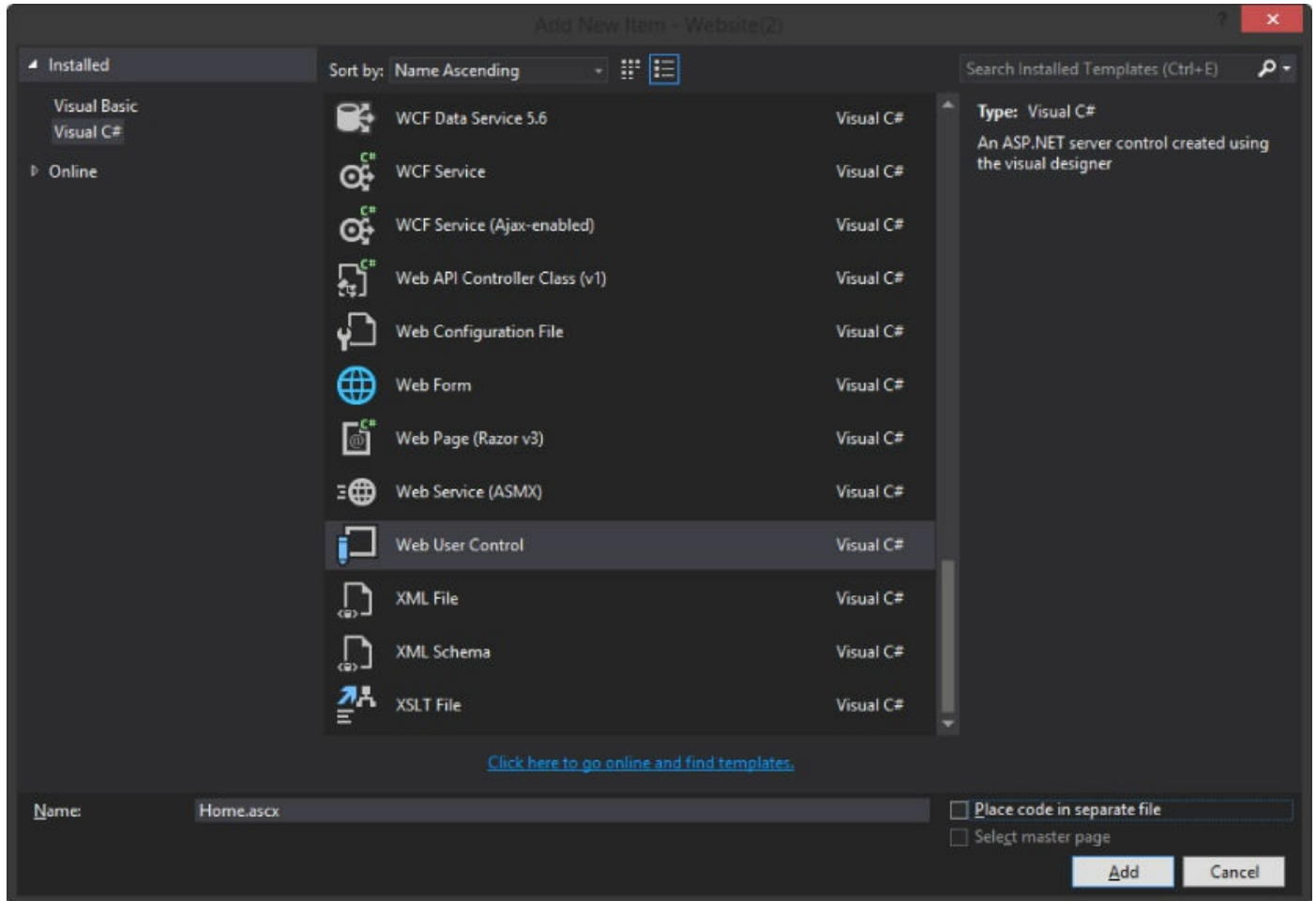


Figure 17.11

Save the file. That's it! You've just created your first DNN skin!

Let's review what you just did. A DNN skin is really just a user control that DNN uses to create your web page layout, which is to say that it is the template for your pages. The first line uses the `Control` directive to specify that this page is a .NET user control, that it uses C# as the language (`vb` is a valid value as well if you plan to add any custom VB.NET code to your skin), and that the skin will inherit from the `DotNetNuke.UI.Skins.Skin` class. Also, `AutoEventWireup` is set to `false`. The third line defines the required content pane for the skin, which we will discuss in the next section.

Basic Layout

Now that you have your skin started, let's add some headers, footers, and panes, and give it some CSS to create a basic layout.

Setting Up a Test Page

Let's first apply the skin to a page so that you can see it start to take shape as you update it. Log in as an administrator or host to the development site where you have been creating your skin files. On the home page of your website, hover over the Edit Page tab in the Control Panel and click the Edit This Page button. Delete all of the modules on the page. Next, hover over the Edit Page tab in the Control Panel and select Page Appearance from the menu. Once the window has opened, scroll down to the Page Skin selection drop-down and expand the options. You will notice quite a few skins listed as `Host:[skin name]`. These are the skins that come preinstalled with DNN. If you created your skin in the `Portals/_default` folder, you should see your skin at the bottom of the list prepended by `Host:.` In this case, the skin is named `Host: SubtleTrend - Home`. Notice that you didn't have to tell DNN anything about this. Once you created a skin folder (`SubtleTrend`) and a skin named `Home.ascx`, DNN recognized this file as a skin. Select this skin and click the Update Page button at the bottom of the window. If you do not see your skin in this drop-down, confirm that you are working within the correct installation, that these requirements have been met, and that your file contains the code from the previous section.

Right now your page doesn't look like much, but you will start working toward changing that.

Content Panes

To provide a place on the page for content to be added through a module, you use content panes. One content pane, called `ContentPane`, has already been added to your page. A content pane is a skin object in a DNN skin that is a placeholder for module content. Specifying content panes in DNN is very simple. To define a content pane, declare a unique ID for a `div` or other HTML element, including `p`, `section`, `header`, `footer`, `main`, `article`, and `aside`, and give it the attribute `runat="server"` to tell the processing server to process it on the server, as it's not traditional HTML. The ID can be whatever you want, as long as it is unique, falls under the guidelines for IDs in HTML,

and contains at least one content pane named `ContentPane`. You don't have to include “pane” in the name, but it is recommended to provide a descriptive ID based on the location/use on the page, such as “sidebar,” “footerpane,” “lowerpane,” and so on.

When you're converting a design to a skin, deciding where to put the content panes is important. Content panes are used to create columns and large sections of content. Many modules can be added vertically in a content pane.

Based on the content pane map for your skin, let's update the code to match the design as shown in [Figure 17.12](#). You're changing out the `div` on the existing content pane for a `main` HTML5 element and adding more layout elements and content panes.

```
<%@ Control language="C#" AutoEventWireup="false"
    Inherits="DotNetNuke.UI.Skins.Skin" %>
<header>
    Logo will go here
    Navigation will go here
</header>
<div id="TopPane" class="top-pane" runat="server" />
<div id="contentWrap">
    <main id="ContentPane" class="content-pane" runat="server" />
    <aside id="AsidePane" class="aside-pane" runat="server" />
</div>
<div id="LowerPane" class="lower-pane" runat="server" />
<footer>
    <div class="footer-wrap">
        <div id="leftFooter">
            Copyright will go here
        </div>
        <div id="rightFooter">
            Register link will go here | login link will go here
        </div>
    </div>
</footer>
```

topPane

REALLY
TRENDY
BANNER

SOME
SUBTLE
TEXT

contentPane

LOREM
& IPSUM

ipsum dolor sit amet, consectetur adipiscing elit. Sed at cursus leo. Nam a iaculis mi. Donec
iputate tempus magna at pellentesque. Curabitur nec nunc dignissim, fermentum ipsum sit amet
arta massa. Aliquam erat volutpat. Curabitur volutpat sit amet nunc nec tempor. Pellentesque
tae ullamcorper elit. Donec tempus sed lorem sed congue. Integer ultrices quis dolor vel
bendum. Donec sed ipsum sed enim pulvinar imperdiet ut ac nisi. Phasellus imperdiet blandit
bendum. Suspendisse condimentum, quam et aliquet dignissim, risus elit eleifend dul, quis matt
it mauris sit amet sem.

id sollicitudin quam nunc, at elementum dolor aliquam vitae. Pellentesque elementum risus
ingue ante sodales, nec semper libero mattis. In scelerisque urna ac sapien rhoncus, et sollicitu
n urna pharetra. Nam ornare fringilla ex at ultricies. Proin euismod aegestas odio, at interdum an
bendum vitae. Vivamus ultrices consequat eleifend. Nullam tristique elementum aliquam. Nam
tae vestibulum orci.

Donec non risus ultricies, elementum risus ac, fermentum dolor. Vestibulum non consequat lorem
Donec vel leo eu dolor posuere vehicula sit amet at justo. Nunc hendrerit suscipit mauris, venena
maximus mi pulvinar accumsan. Vestibulum id mi ipsum. Phasellus non tempus justo. Vestibu-

asidePane

Serif on
the *<aside>*

Lorem ipsum dolor sit amet, consec-
tetur adipiscing elit. Sed at cursus
leo. Nam a iaculis mi. Donec vulpu-
tate tempus magna at pellentesque.
Curabitur nec nunc dignissim, fer-
mentum ipsum sit amet, porta massa.
Aliquam erat volutpat. Curabitur
volutpat sit amet nunc nec tempor.
Pellentesque vitae ullamcorper elit.
Donec tempus sed lorem sed congue.
Integer ultrices quis dolor vel
bibendum. Donec sed ipsum sed
enim pulvinar imperdiet ut ac nisi.
Phasellus imperdiet blandit
bibendum. Suspendisse condimen-
tum, quam et aliquet dignissim, risus
elit eleifend dul, quis mattis elit
mauris sit amet sem.

lowerPane

Proin vitae fringilla felis. Sed tincidunt erat massa, at conse
Fusce vel turpis ac est varius sodales. Aenean lacus orci, facilisis at ligula id, elementum porttitor odio. Integer orci neque, pulvinar nec justo
a, mollis semper leo. Donec consectetur orci eget pulvinar gravida. Nullam sem mauris, fringilla semper lectus a, euismod commodo quam.
Duis at fermentum augue, et euismod augue. Sed commodo fermentum viverra. Donec tempor elementum nunc, sit amet cursus sem pel-
lentesque at. Phasellus tristique orci in purus sodales cursus. Curabitur vel rhoncus quam, a sodales mauris.

Fusce nec ornare ligula. Duis ac suscipit nisi. Sed eget nisi sit amet enim cursus rhoncus. Vivamus id consectetur nisi. Mauris fribus urna
sit amet quam presum, non blandit erat efficitur. Donec arcu ante, sagittis sit amet est a, elementum commodo diam. Ut convallis, urna nec
fringilla auctor, lectus ligula rhoncus purus, ac accumsan lectus dolor eu neque. Mauris tempor du, sit amet est ultrices, sit amet ullamcorper
metus fribus. Sed vulputate tellus nulla. Integer porta magna quis condimentum mollis. Sed tincidunt ut magna nec consequat. Nulla con-
vallis, tellus at pretium pretium, ante orci maximus massa, id dictum eros sapien in massa. Donec imperdiet, enim id porttitor congue, ipsum
nisi sagittis erat, eu ultrices purus nisi quis libero. In hac habitasse platea dictumst. Etiam tempor molestie risus. Quisque eu nisi justo.

Figure 17.12

You have added `header` and `footer` elements with some text as placeholders. You also created a few new content panes by setting the `runat` attribute to `server` and giving them the unique IDs as follows: `TopPane`, `AsidePane`, and `LowerPane`. (Note that versions prior to DNN 7.3.0 do not support using HTML5 elements as content panes. You will need to use a `div` as a content pane inside of the desired HTML5 element.) You have also made the content panes be self-closing elements. This is an optional, but helpful, technique when creating your skin to be able to visually separate a content pane from a standard element. Because this will be converted to a content pane at the server runtime, you will never be left with a self-closing element once rendered to the page. You also wrapped the `ContentPane` and the `AsidePane` in a `div`, which will allow you to keep your content centered on the page as defined by the design.



NOTE

It is good practice to apply a class such as `aside-pane` to your content panes. This is because DNN prepends `dnn_` to the ID you specify on the content pane upon rendering to the page. So, specifying `AsidePane` in your HTML will render the ID on the content pane as `dnn_AsidePane`. However, the class name you assign will not change.

Adding Test Content

After getting some content panes on your skin, now is a good time to add some filler content to the test page so that you can see the effects of the changes to the layout as you make them. Once you have added some content, you will continue styling your skin.

Adding HTML Modules to the Page

While logged in as an administrator or host, navigate to the home page that you have been editing. Then hover over the Modules tab in the middle of the Control Panel at the top of the page and click on the Add New Module link. From the drop-down, select Common and the list of modules will filter to show only the HTML module. Click and drag the module to the `ContentPane`. Once that module has been added, add a second module to the page in the Aside Pane by hovering over HTML module in the list of modules and then hovering over the crosshairs and selecting Add to AsidePane. You just added new text modules to the page in two different ways: dragging and dropping and selecting the pane through the add-to-pane drop-down option. Add an HTML module to the TopPane and one to the LowerPane. Once you're finished, you can hide the Add New Module panel by clicking the Cancel link directly below the Edit Page tab. Notice that these modules are stacked vertically and not as the design specifies. This is because you have not applied any CSS to the skin. We will walk through laying these out in the CSS section.

Updating Module Titles

Hover over the title of the first module where it says Text/HTML and click the pencil icon. This puts the module into a quick edit mode. Now, change the title to be “Really Trendy Banner” and click the save icon. On the second

module that is still labeled “Text/HTML,” hover over the gear icon and select Settings from the drop-down menu. Once the settings pop-up appears, change the text for Module Title (on the Module Settings tab in the Basic Settings section) to `Lorem
& Ipsum`. Notice that you are able to place HTML into the module's title. This will allow you to be creative in making reusable, flexible styles for your module titles. Scroll down to the bottom of the pop-up and click the Update button. Update the next module through the module settings window to `Serif on the <aside>`, which will render to the page as `Serif on the <aside>`. Lastly, update the bottom module's title to “Lower Content.”

Adding Text to Your Modules

DNN automatically adds some content to the HTML module that is added to the page. However, the content is there to provide guidance for editing the module. If you were to log out, you would not even be able to see the modules on the page. They remain hidden to site visitors until the content has been updated. Similar to the previous step of editing the title of a module, there are two ways to edit content for a module: a quick edit and a more detailed approach. However, when editing content to the HTML module, it is best to avoid the quick edit as it can introduce some styles and elements that will not be desirable on your page. On the module in the `TopPane`, labeled `Really Trendy Banner`, hover over the Edit Content icon next to the Settings icon in the top right of the module and select the Edit Content option. Once the Edit Content pop-up has appeared, select the HTML tab at the bottom of the body on the Rich Text Editor. This switches the editor to allow for adding HTML content. (Note that any JavaScript added into this editor will be removed upon saving.) Paste the following content as specified by the design: `<p>Some subtle text</p>`. In the remaining HTML modules, paste some sample content into the body of the editor for each module. You can get some sample text from your favorite Lorem Ipsum generator. This example uses the following:

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec
maximus pulvinar
mollis. Maecenas consectetur, risus pretium suscipit finibus, leo leo
maximus
diam, nec mattis velit turpis sit amet mi. Aenean dictum ex ut augue
hendrerit
tempor. Maecenas ut lobortis ex. Aliquam luctus sem mi. Proin non
ante et augue
venenatis egestas consequat ut tortor.</p>
```

```
<p>Nulla placerat varius odio, ut lobortis justo dictum at. Praesent  
nisi neque,  
placerat in dolor sed, condimentum malesuada sem. Pellentesque  
vulputate eros ut  
imperdiet maximus. Curabitur varius quam in dolor semper auctor. Nunc  
ligula  
risus, molestie sit amet magna dapibus, pharetra consectetur ligula.  
Maecenas sed  
ex id nibh lobortis lacinia.</p>  
<p>Etiam porttitor orci id ultricies rhoncus. Ut id neque ac enim  
pellentesque  
vestibulum ut eu ante.</p>
```

Scroll down and click the Save button. Repeat with the Aside Pane and Lower Pane. You now have some content on your page that will allow you to see the impact of your work as you progress.

CSS

Now that you have the basic HTML set up, you can start adding some style to the skin. In Visual Studio, right-click the skin folder called `SubtleTrend` and select **Add ➔ **Add New Item**. Select **Style Sheet** from the options listed. Name it `skin.css` and click the **Add** button. Let's give the pages some color, center the content, and place the content panes as designed by replacing the contents of the file with the following CSS:**

```
/* Gives selected content color and background color */  
#Body *::selection{  
    background-color:#D84D41;  
    color:#15313F;  
}  
#Body{  
    background-color:#1F4B59;  
    color:#FFEDCB;  
}  
/* Centers header on page and centers content vertically with line-  
height */  
header{  
    margin:0 auto;  
    width:1170px;  
    line-height:120px;  
}  
/* clearfix trick to clear nested floats */  
header:after {  
    content: "";  
    display: table;  
    clear: both;  
}
```



```

/* Set page width and center main content section */
#contentWrap{
  margin:45px auto 0;
  width:1170px;
}
/* clearfix trick to clear nested floats */
#contentWrap:after {
  clear: both;
  content: "";
  display: table;
}

/* Set width of main content to about 65% and float left */
#contentWrap .content-pane {
  float:left;
  width:760px;
}

/* Set width of aside pane to about 33%
   to allow padding between panes and float right */
#contentWrap .aside-pane{
  float:right;
  width:380px;
}
/* Clears float, centers content, and gives
   spacing above the lower content pane */
.lower-pane{
  clear:both;
  margin:45px auto 0;
  width:1170px;
}
/* Sets color and size for full width footer */
footer{
  background-color:#D94D41;
  clear:both;
  color:#FFEDCB;
  height:90px;
  line-height:90px;
  margin-top:45px;
}
/* Centers content for footer */
.footer-wrap{
  margin:0 auto;
  width:1170px;
}
/* clearfix trick to clear nested floats */
footer:after {
  clear: both;
  content: "";
  display: table;
}

```

```

}

/* Style links within footer */
footer .footer-links {
    color:#FFEDCB;
}
footer #leftFooter{
    float:left;
    width:45%;
}
footer #rightFooter{
    float:right;
    text-align:right;
    width:45%;
}

```

Save this file and refresh your test page on your site. Notice that your skin picked up your CSS file automatically! Sure, it's not all that pretty yet, but you can fix that later. If you view the source of the rendered page and scan down the code for a link to `skin.css`, you will notice that, in fact, your CSS file has been added to your skin for you. How did this happen? The skinning engine that comes with DNN has some very handy features built into it. Any CSS file named `skin.css` will be automatically added to all of the skins within that particular skin package. Next, let's create a new CSS file in the skin folder following the same steps as the `skin.css`, but this time, name it `home.css`. Add the following CSS to this file, save it, and then refresh your page:

```

#Body {
    background-color:#000;
}

```

There again, DNN updated the page to include your new CSS file. This is another feature of the powerful skinning engine. Any CSS file named the same as a skin file will be automatically linked in the skin (such as `two-column-with-footer.ascx` and `two-column-with-footer.css`). This CSS file loads after the `skin.css` file and will take priority over the `skin.css` file. This is helpful when you need to override to a specific skin such as a background image on the `body` element. You can delete this file as you do not need any skin-specific styles.

Although it is helpful that DNN linked to the CSS automatically, you need to understand a little more about where in the document that link shows up. As CSS by name is cascading, the order that CSS gets loaded is very important. Also, you may notice that some styles are being specified without being

defined in the stylesheet, such as those from `default.css`. DNN has several stylesheets that get loaded onto the page. They are as follows:

- `Default.css`
- `Module.css`
- `Skin.css`
- `[SkinName].css`
- `Container.css`
- `[ContainerName].css`
- `Portal.css`

While creating the styles for your skin, realize that any of these files could have an impact on your styles and you may need to override them in your skin to make sure your design is displayed correctly. For example, the `default.css` stylesheet specifies font colors, sizes, and line heights for various content within modules. `Default.css` is part of the core of DNN and has some fallback styles to make sure that content is presentable even when there are no predefined styles for a skin. It is recommended that you not edit this file, as it can be overridden when upgrading to a different version of DNN.

The next CSS files to load are the files that are associated with any modules that have been loaded onto the current page. These files are set up by the module's developer and could have an impact on your specified styles as well. It is also not a good idea to edit these files as upgrading the module could overwrite any changes you have made.

Following the `module.css` files are the skin stylesheets that you create for your skin, including the skin's default stylesheet (`skin.css`). Also, any specific stylesheets for containers used on the current page are added. Lastly, the `portal.css` stylesheet is loaded. This is a stylesheet that is edited through the Site Settings when logged in as a site administrator. It is recommended to not use this file for any skin-related styling, as site administrators would be able to edit and possibly remove styles, which could break the skin's design.

What if you want to include some other CSS such as for a CSS responsive framework or a JavaScript plugin? For this, you can leverage some skin objects that you'll read about in a later section. See the “Skin Objects” section later in this chapter for more information.

Overriding Basic Styles

Because DNN loads the `default.css` file with some predefined styles, you must add a few more styles in the `skin.css` file to make sure that the style persists. You may have noticed that you defined the color on the `body` tag at the beginning of the CSS. You do that by using the `#Body` ID instead of the `body` element. DNN adds this ID to the `body` element, as there can be conflicting styles in the rich text editor, depending on the text editor being used in DNN. This allows you to target the body of the page and not the body tag in the WYSIWIG editor. You may wonder why the text color didn't apply to the content of the page. Many DNN modules add a class of `Normal` to the content pane within the container for the module. DNN specifies a color of `#444` in the `.Normal` definition in `default.css`. This needs to be overridden in the `skin.css` file. Add the following CSS to the end of your `skin.css` file to address a few of the styles defined in `default.css`:


```
/* Page styles */
.Normal{
    color:#FFEDCB;
    font-family:Arial, Helvetica, sans-serif;
    font-size:18px;
    line-height:1.67;
}
h1,h2,h3,h4,h5,h6{
    color:#FFEDCB;
    font-weight:normal;
}
h2, h2 .title{
    font-family:Georgia, 'Times New Roman', Times, serif;
    font-style:italic;
    font-size:48px;
}
```

Linking to Files in Your Skin

You may have noticed that you are using HTML5 and since DNN 7 supports IE8 and up, you have an issue. IE8 does not support HTML5 elements! Thankfully, the web development community has provided a shim, or “shiv” as it is known, in the form of `html5shiv`. `Html5shiv` is a very popular fix for forcing IE8 to recognize HTML5 elements. You need to download this file and reference it in your skin. Download the HTML5Shiv here:

<https://github.com/aFarkas/html5shiv>.

After downloading the shiv, right-click the `SubtleTrend` skin folder and select

Add  Folder. Name the folder `js`. Place the `html5shiv.min.js` file that was downloaded into this folder and add the following code above the `header` element in `Home.ascx`:

```
<!--[if IE 8]>
  <script src="<%=SkinPath %>js/html5shiv.min.js"></script>
<![endif]-->
```

When DNN renders your skin onto your page, all links to resources included are made relative to the page that you are viewing. For example, if you had added the link as relative to the skin such as `<script src="js/html5shiv.min.js"></script>` and navigate to an About page in the root of your site with the URL <http://yoursite.com/about>, the link rendered for your JavaScript file would be <http://yoursite.com/About/js/html5shiv.min.js> and the link to the file would be broken. The actual location of the file is http://yoursite.com/Portals/_default/Skins/SubtleTrend/js/html5shiv.min.js.

You could create a link directly to the file including the `/Portals/_default/Skins/SubtleTrend` path, but, what would happen if this skin was installed into a specific site, and not in the `_default` folder? The link would need to be updated. Thankfully, DNN provides `SkinPath`, a property of the skin class that allows you to always be relative to the skin, independent of the site's folder or DNN installation. Placing this at the beginning of your path will then render the link to the file correctly.

This technique is typically used more when you need to include an image in the skin. Note that any references to images in CSS are relative to the CSS file within the skin, so this is not an issue for CSS. For an image located in an images folder within your skin package folder, you would write your code in the following manner:

```

```

Document Setup

Your page is starting to take shape, but you might have noticed that with the ASCX approach, you did not write some of the basic elements that make up a valid web page within the skin file. You have not declared the document `doctype`, nor created the `head` or the `body` of the document. You may have also noticed that when you looked at the HTML approach, DNN stripped these out of the skin when it was parsed. This is because DNN automatically adds all of the elements that you need to have on your page when the page is rendered. DNN skins only include the contents of the `body` of the page.

However, there are some instances when you'll want to change some elements that are defined by DNN, such as `DOCTYPE`, or add some elements, such as `meta` tags and additional CSS and JS files. `meta` tags and CSS and JS files are configured via configuration files and skin objects, which will be discussed in the coming sections.

DOCTYPE

Since you are using HTML5, you need to specify that you are using an HTML5 `DOCTYPE`. Even though you don't have direct access to the document `DOCTYPE`, DNN provides a way for you to accomplish this. DNN uses a specific XML file with the `DOCTYPE` specified in the contents of the file. First, you will create a new XML file by right-clicking the root folder of the new skin `SubtleTrend` and selecting Add ➤ Add New Item. Choose XML File as the file type and name it `skin.doctype.xml`. DNN will recognize this file and use its contents to add a `DOCTYPE` to all skins located within this skin folder. Replace the contents of the file with the following code:

```
<SkinDocType><![CDATA[<!DOCTYPE html>]]></SkinDocType>
```

This file can also be found in the preinstalled skin, called Gravity. Just copy and paste this file into your skin folder. To specify a different `DOCTYPE`, replace the contents of the `CDATA[...]` in the file with your desired `DOCTYPE` declaration.

A fallback `DOCTYPE` can also be set through the DNN host settings. If no `DOCTYPE` is specified in the skin package, DNN will fall back to the specified `DOCTYPE`, which is set in the Host settings. Log into your site as a host, navigate to Host ➤ Host Settings, and expand the Appearance section. There you will see an option to set a fallback skin `DOCTYPE`. Although this option can be used to set a `doctype`, it is good to create the `skin.doctype.xml` file, as

setting the fallback skin `DOCTYPE` will impact all sites on this DNN installation. Setting this file will also help you when moving or installing the skin package to another installation where you do not have control of this setting.


Skin Objects

Skin objects are user controls that provide a prepackaged functionality that can be reused as needed. Skin objects will often allow for display of dynamic data to the web page or some other functionality that could not be accomplished in the skin alone. Several are included with DNN, with several other third-party modules available. Uses range from displaying a logo to a login link to linking to a JavaScript file or displaying a menu.

Registering Skin Objects

To use one of the preinstalled skin objects, two things are required in the skin. The first is to register the control in the skin, and the second is to define the skin objects and their parameters. For example, to use the Logo skin object, you register the control in the skin by adding the following code to the `home.ascx` file. Place it below the `<%@ Control ... %>` directive:

```
<%@ Register TagPrefix="dnn" TagName="LOGO"
Src="~/Admin/Skins/Logo.ascx" %>
```

The `TagPrefix` attribute will assign a prefix that will be used in the skin to associate the user control. The `TagName` attribute will associate the name with the user control. So, in this case, you would start the skin object as `<dnn:LOGO />`. The source attribute points to the user control that you are using, which in this case is located in the `Admin` (at the root of the install)  `Skins` folder. Navigating to this folder in your DNN installation will show you all of the skin objects that come prepackaged with DNN.

Defining Skin Objects

To use the skin object in your skin, place the skin object control in your skin where it is intended to be rendered, specifying an ID and setting the `runat` attribute equal to `server`.

```
<dnn:LOGO id="siteLogo" runat="server" />
```


This is all that is required to place the site logo image and add a link to the home page in your skin. Let's see where this goes in your skin.

```
<%@ Control Language="C#" AutoEventWireup="true"
Inherits="DotNetNuke.UI.Skins.Skin" %>
<%@ Register TagPrefix="dnn" TagName="LOGO"
Src="~/admin/Skins/logo.ascx" %>
```



```
<header>
  <dnn:LOGO ID="siteLogo" runat="server" />
  Navigation will go here
</header>
```

...

If the logo doesn't show up on your page, this might be because you have not defined a logo for your website. Log in to your site as the administrator or host and navigate to Admin  Site settings. On the Basic Settings tab, expand the Appearance section and see if a file is specified for the logo. If not, upload a new image file for a logo. Scroll to the bottom of the page and click the Update button. Navigate back to your test page; you should now have a logo at the top of the page.

Let's explore what this skin object rendered in the HTML:

```
<a title="My DNN Site" id="dnn_siteLogo_hypLogo"
href="http://dnndev.me/">
  
</a>
```

As you can see, the skin object produced the image that you specified as the site logo in the Site Settings with a link back to the root of the site (my local URL that is set up in IIS is `dnndev.me`). Notice that it also used the name of the site (also specified in Site Settings) as the `alt` attribute for the image and the `title` attribute of the link. DNN also created IDs for each element using the ID that you specified in the skin object while prepending `dnn_` and appending `_hypLogo` and `_imgLogo` to the link and image, respectively. This will provide you with the ability to apply CSS to the elements without having to wrap the skin object in an extra element.


Next, you will add login and register links and a copyright to the footer of the site. First, to register the skin objects, add the following code below the existing skin object registration code:

```
<%@ Control Language="C#" AutoEventWireup="true"
  Inherits="DotNetNuke.UI.Skins.Skin" %>
<%@ Register TagPrefix="dnn" TagName="LOGO"
Src="~/admin/Skins/logo.ascx" %>
<%@ Register TagPrefix="dnn" TagName="LOGIN"
Src="~/admin/Skins/login.ascx" %>
<%@ Register TagPrefix="dnn" TagName="USER"
Src="~/admin/Skins/user.ascx" %>
<%@ Register TagPrefix="dnn" TagName="COPYRIGHT"
```

```
Src="~/admin/Skins/copyright.ascx" %>
```

Next, define the skin objects in the footer:

```
...
<footer>
  <div class="footer-wrap">
    <div id="leftFooter">
      <dnn:COPYRIGHT ID="siteCopyright"
        CssClass="copyright-text"
        runat="server" />
    </div>
    <div id="rightFooter">
      <dnn:USER ID="siteRegistration"
        LegacyMode="false"
        ShowAvatar="true"
        ShowUnreadMessages="true"
        CssClass="footer-links"
        Text="Register for Site"
        runat="server" /> |
      <dnn:LOGIN ID="siteLogin"
        CssClass="footer-links"
        LegacyMode="false"
        Text="Sign In"
        LogoffText="Sign Out"
        runat="server" />
    </div>
  </div>
</footer>
```

In this example, you can see that there are several different attributes available for the skin objects. Each skin object can have its own set of attributes. If you are using Visual Studio, you may have noticed that these were shown as available attributes in the skin objects by IntelliSense. This is one of the big reasons why using Visual Studio as the code editor can be very helpful. The COPYRIGHT skin object displays the text from the Copyright text field located on the Admin  Site Settings page in the Basic Settings tab in the Site Details section. For this skin object, you can set a class for the text that is produced by the skin object through the use of the `CssClass` attribute. The HTML that is produced for the Copyright skin object is as follows:

```
<span class="copyright-text" id="dnn_siteCopyright_lblCopyright">
  Copyright 2014 by DNN Corp
</span>
```

DNN added a `span` element with the class that you specified in the skin object

as well as prepending the ID you specified with `dnn_` and appending `lblCopyright`. A handy feature that DNN uses for the copyright is keeping the current year in the copyright by replacing the year in the Copyright text field in the Site Settings page with `[year]`.

Next, you will look at the Login skin object. This skin object has even more attributes. The `Text` attribute shows the text of the link that is rendered while logged out, and the `LogoffText` attribute shows the text of the link while logged in. The Login and User skin objects both define a `LegacyMode` attribute. Setting this attribute to `false` will allow you to leverage some of the social aspects of DNN, such as message and site notifications and user avatars. If you save your skin and refresh your page, you will see that all of these are rendered out, yet they are unstyled. Now that you have explored the advanced functionality of these skin objects, you can use much more simplified versions since the design does not require this level of complexity.

If you would like to leverage the updated versions of these, you can find the CSS for these controls in `skin.css` for the Gravity skin, which is located in the Portals `_default\Skins\Gravity` folder. You can simplify the Login and User skin objects by using the legacy mode. Update the skin objects as follows:

```
<footer>
  <div class="footer-wrap">
    <div id="leftFooter">
      ...
    </div>
    <div id="rightFooter">
      <dnn:USER ID="siteRegistration"
        CssClass="footer-links"
        Text="Register for site"
        LegacyMode="true"
        runat="server" /> |
      <dnn:LOGIN ID="siteLogin"
        CssClass="footer-links"
        Text="Log In"
        LogoffText="Log Out"
        LegacyMode="true"
        runat="server" />
    </div>
  </div>
</footer>
```

Set `LegacyMode` to `true` for the User and Login skin objects. You also removed the attributes for `ShowAvatar` and `ShowUnreadMessages` from the User skin

object, which will default to `false`. Now this will render links with the classes specified in the `CssClass` attributes and links to the pop-ups for site registration and login forms when logged out. However, when you're logged in, DNN switches these with a link to the user's profile page for the User skin object and a link to log out of the site for the Login skin object. As you can see, advanced coding would be required to perform all of these functions were these skin objects not available.

Setting Up a Responsive Web Design Skin

Now that you have a basic layout, let's make this site ready for the different sized devices that will access it. Enabling your site to be displayed properly on all devices takes some planning to ensure that your users will be able to easily navigate your site from whatever device they are using. It is a good idea to work through the layout with wireframes for the different screen sizes before you begin designing the site.

Once you have decided how you are going to handle the layout for different media sizes, you are ready to make your skin responsive. There are two things required for this: setting up the viewport and providing CSS for the other sized media devices.

Setting the Viewport Using the Meta Skin Object

Setting up the viewport to device width will cause the site to scale correctly to the size of the device; it won't try to zoom out to include the width of the whole page. The initial scale setting will establish the correct ratio between the device and CSS pixels. To set up the viewport, you must add the following `META` tag to the `head` of the document:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Yet, you do not have access to the document's `head` element. However, DNN provides the `Meta` skin object to handle this. First, register the skin object to the page by adding it to the list of the skin object's registration declarations.

```
<%@ Register TagPrefix="dnn" TagName="META"
Src="~/admin/Skins/Meta.ascx" %>
```

Directly below the registration declarations, define the skin object as follows:

```
<dnn:META ID="viewport"
```

```
Name="viewport"
Content="width=device-width, initial-scale=1.0"
runat="server" />
```

Note that you are setting the attributes for `name` and `content` the same as you would in normal HTML. The main difference is that you are writing this as a skin object, which will insert the appropriate `META` tag to the `head` of the document (however, if you want to set the `http-equiv` attribute, you'll need to use `HttpEquiv`, without a dash).

Creating the CSS Using the Styles Skin Object

Next, you need to create some CSS to define how the browser should handle the page layout at a smaller size. You can do this two different ways. You can specify `@media` queries within the `skin.css` files or you can create a specific CSS file for each size and link directly to those files. For this example, you will create specific CSS files for each skin size.

Start by creating a stylesheet for smaller desktops or large tablets that are 1200 pixels wide or smaller. Even though this includes small tablets and phone sizes, you'll create two more new stylesheets a bit later to address those sizes specifically. Add a new file to your skin package by right-clicking your skin folder, called `SubtleTrend`, and selecting `Add` → `Add New Item` from the options. Select `Style Sheet` from the options and name it `media-medium-screens.css`. Add the following CSS to the file and save it:

```
/* Reduce height of header */
header {
    line-height:90px;
}

/* Reduce width of full width elements to fit smaller size */
header,
#contentWrap,
.lower-pane,
.footer-wrap{
    width:960px;
}

/* set width of aside pane to about 65% */
#contentWrap .content-pane {
    width:625px;
}

/* set width of aside pane to about 33% to allow padding between
panes */
```

```
#contentWrap .aside-pane {
    width:320px;
}
```

To link directly to the CSS file while keeping it relative to the skin package, you can leverage the Styles skin object. Register the Styles skin object with your other skin object registration declarations and define it in your skin. (Note: order does not matter when registering skin objects.)

```
<%@ Register TagPrefix="dnn" TagName="STYLES"
Src="~/admin/Skins/Styles.ascx" %>
```

Add the following code below the Meta skin object:


```
<dnn:STYLES id="mediaMediumScreens"
    Name="mediumScreens"
    Media="only screen and (max-width: 1200px)"
    StyleSheet="media-medium-screens.css"
    UseSkinPath="true"
    runat="server" />
```

Now refresh your page and resize your browser to a size less than 1200px wide. The content width on the page will reduce in size to 960 pixels, as defined in the CSS for displays smaller than 1200px.

This created the following link to the CSS file in the `head` of the document:

```
<link id="mediumScreens"
    href="/Portals/_default/Skins/SubtleTrend/media-medium-
screens.css"
    rel="stylesheet" type="text/css"
    media="only screen and (max-width: 1200px)">
```

So, let's take a look at what you just did. The `Name` attribute is used as the ID for the link. The `StyleSheet` attribute specifies the stylesheet to link to. Setting the value of the `UseSkinPath` attribute to `true` causes the stylesheet you specified to be relative to the skin package of the current skin. Declaring this as `false` would set the URL of the `StyleSheet` attribute relative to the root of the site. The `Media` attribute specifies the type of device and the maximum browser width that the CSS file should take effect.

Next, let's create a layout for smaller size screens such as large tablets in portrait mode and small tablets less than 992px. Create a new CSS file by right-clicking the `SubtleTrend` skin package folder and selecting Add  Add New Item from the options. In the Add New Item window, select Style Sheet and name it `media-small-screens.css`.

Place the following CSS in the file and save it:

```
/* Reduces size of logo and pushes it away from left edge of screen
*/
#dnn_siteLogo_hypLogo {
    max-width: 300px;
    margin-left:30px;
}

/* set width of all panes and page elements to fluid width */
header,
#contentWrap,
#contentWrap .content-pane,
#contentWrap .aside-pane,
.lower-pane,
.footer-wrap{
    width:100%;
}
/* Add padding to footer */
footer {
    padding:0 30px;
}
```

Since you have already registered the Styles skin object, you only need to define the new instance of the skin object for smaller screens. Add the following code to the `home.ascx` file, below the definition for the Styles skin object for the medium screens:

```
<dnn:STYLES id="mediaSmallScreens"
    Name="smallScreens"
    Media="only screen and (max-width: 992px)"
    StyleSheet="media-small-screens.css"
    UseSkinPath="true"
    runat="server" />
```

The order that your styles are defined in your skin is important. The stylesheets are loaded in the order they are defined in your skin. The stylesheet that is defined last will take precedence as long as the conditions set by the `Media` attribute are `true`. Also, there is another attribute, `IsFirst`, that can be set to `true` to make sure that file is loaded before any of the other stylesheets. However, if multiple files with the attribute `IsFirst` are set to `true`, they will be loaded in the order they are defined within the skin. The default value for `IsFirst` is `false`.

Refresh your page and reduce your browser width to less than 992px. The content in the content pane should now be full width and the content in Aside Pane should now fall below the content pane content. Note that you have

made only a few changes to override the medium screen styles. Because you set up the medium stylesheet to be a maximum width of 1200px, that stylesheet is also inherited at the smaller sizes and only the styles defined in the smaller stylesheet will override.

Lastly, you will create a new stylesheet called `media-xtra-small-screens.css`. Place the following styles in the stylesheet and save it:

```
/* Reduce size of logo to fit smaller screens */
#dnn_siteLogo_hypLogo img{
    max-width: 170px;
}

/* Change footer height to be auto to fit the content */
footer {
    height: auto;
    line-height:25px;
    margin-top: 0;
}
/* Left align text and set Register and Login links to fall below
copyright */
footer #leftFooter,
footer #rightFooter {
    text-align: left;
    width: 100%;
}
```

Then link to your new file containing the Styles skin object using the following code:

```
<dnn:STYLES id="mediaXtraSmallScreens"
    Name="xtraSmallScreens"
    Media="only screen and (max-width: 768px)"
    StyleSheet="media-xtra-small-screens.css"
    UseSkinPath="true"
    runat="server" />
```

This didn't do much other than reduce the size of the logo and cause the content of the footer to stack, but as you begin creating containers, you will be adding more CSS to these files.

Conditionals for Internet Explorer

As mentioned previously, DNN officially supports Internet Explorer versions 8 and up. Although your skin doesn't need it, as the CSS already falls back gracefully in IE8, the Styles skin object is also helpful for adding conditionals for Internet Explorer stylesheets. For example, to load a stylesheet for IE

versions less than or equal to 8, set the `Condition` attribute equal to `IE lte 8`. For example, the following code:

```
<dnn:STYLES id="IE8styles"
  Name="IE8styles"
  Condition="IE 8"
  StyleSheet="IE8.css"
  UseSkinPath="true"
  runat="server" />
```

would render the following:

```
<!--[if IE 8]>
  <link id="IE8styles" rel="stylesheet"
  type="text/css"
  href="/Portals/_default/Skins/SubtleTrend/IE8.css" />
<![endif]-->
```

Available Skin Objects

There are many skin objects included with DNN that we are not able to fully describe in this chapter. To see the available skin objects for DNN 7, refer to the DNN Wiki on the DNN Software website.

Client Resource Management (CRM)

The speed of your website is important, and every millisecond counts, especially when dealing with mobile devices. Multiple calls to the server for CSS and JavaScript files take time and add up. Search engines have placed a lot of importance on this in regard to their page rankings. DNN provides the Client Resource Manager, which you can leverage to merge all of your multiple CSS files and JavaScript files into one call each from the server.

The Client Resource Manager will take only the files that need to be loaded for each specific page and determine if it is already being called to avoid duplication, which can also lead to conflicts in JavaScript. It will then combine all of the files and cache them on the server. The Client Resource Manager, by default, is not enabled to compress and combine all files; however, you can still use it to link to CSS and JavaScript files for your skin. To learn more about the Client Resource Manager, see [Chapter 4](#).

To make use of the CRM in the skin, you must first register the control as follows:

```
<%@ Register TagPrefix="dnn"
```

```
Namespace="DotNetNuke.Web.Client.ClientResourceManagement"  
Assembly="DotNetNuke.Web.Client" %>
```

The CRM can now be used by the `DnnCssInclude` and `DnnJsInclude` skin objects.

CSS

Let's say, for example, that you want to leverage a CSS framework to assist with layout and flexibility for mobile device resolutions. To load a CSS file through the Client Resource Manager, add the following code below the `Register` definitions:

```
<dnn:DnnCssInclude FilePath="myOtherCSS.css" runat="server"  
  PathNameAlias="SkinPath" Priority="40" />
```

This will place a link to the CSS file in the `head` of the document. The `FilePath` attribute defines the link to be relative to the skin package. The `Priority` will specify where it will be loaded relative to other resources (see [Table 17.1](#) for the default DNN CSS priorities). For example, if you wanted to load the file just before `skin.css` is loaded, you would specify the value for the priority attribute between 11 and 14. If you wanted to load the file last, you would specify a priority of greater than 35. In the previous example, the linked to file would be placed after all of the other CSS files.

Table 17.1 CSS File Priorities

CSS File	Priority
DefaultPriority	100
DefaultCss	5
AdminCss	6
FeatureCss	7
IeCss	8
ModuleCss	10
SkinCss	15
SpecificSkinCss	20
ContainerCss	25
SpecificContainerCss	30
PortalCss	35

JS

Adding a JavaScript file to the CRM is very similar to adding a CSS file. Create a new folder in the skin's folder and name it `js`. Then, add a new JavaScript file to the `js` folder and name it `scripts.js`. Once the Resource Manager has been registered in the skin, you can load the file using the following code:

```
<dnn:DnnJsInclude FilePath="js/scripts.js" runat="server"  
  PathNameAlias="SkinPath" Priority="50" />
```

This will link to the `scripts.js` file located in the `js` folder within the current skin package. [Table 17.2](#) shows a list of the priorities of the JavaScript files loaded in by DNN.

Table 17.2 JavaScript File Priorities

JavaScript File	Priority
DefaultPriority	100
jQuery	5
jQuery Migrate	6
jQuery UI	10
DnnXml	15
DnnXmlJsParser	20
DnnXmlHttp	25
DnnXmlHttpJsXmlHttpRequest	30
DnnDomPositioning	35
DnnControls	40
DnnControlsLabelEdit	45
DnnModalPopup	50
HoverIntent	55

Another useful attribute for JavaScript is the `ForceProvider` attribute. By specifying a provider, you can dictate where in the document the file will load: in the `head` or in the top or bottom of the `body` element. `DnnBodyProvider` is the default provider specification for JavaScript files.

The following providers are available and will be useful with JavaScript:

- **DnnPageHeaderProvider:** Places the link to the file in the `head` of the web page document
- **DnnBodyProvider:** Places the link to the file at the top of the `body` of the document
- **DnnFormBottomProvider:** Places the link to the file at the bottom of the `body` of the document

Using the DNN's CRM, let's change the `html5shiv` that you added earlier to use the CRM. Remove the following code:

```
<!--[if IE 8]>  
  <script src="<%=SkinPath %>js/html5shiv.min.js"></script>  
<![endif]-->
```

And replace it with the following:

```
<dnn:DnnJsInclude FilePath="js/html5shiv.min.js" runat="server"  
  PathNameAlias="SkinPath" Priority="50"  
  ForceProvider="DnnPageHeaderProvider" />
```

Notice that you are no longer using the `<%=SkinPath%>` property, as `SkinPath` is already available in the skin object.

Navigation

DNN has a very powerful page management system. It allows pages to be added, removed, renamed, hidden, and disabled, with all of this action being dynamically displayed in the navigation. This page structure is rendered to the skin through the use of a navigation provider. DNN is flexible enough to allow for several different navigation providers; however, the default navigation provider included with DNN, called DDR Menu, provides a great deal of flexibility through custom templates.

Sample Menu Structure

Let's add the following sample HTML, which represents the intended rendered navigation for the site to the skin. Then, you will add CSS to style the menu as desired. Once you have the HTML styled appropriately, you will then create the template that the DDR Menu provider will use to dynamically create the navigation based on the page structure from the site. The DDR Menu provider template engine is flexible enough to take the approach of creating the desired HTML and reverse engineering it into a menu template. Replace the content in the header after the logo skin object with the following HTML:

```
...
<header>
  <dn:LOGO ID="siteLogo" runat="server" />
  <a href="#" id="menuToggle">Menu</a>
  <nav>
    <ul>
      <li class="first"><a href="#">Home</a></li>
      <li class="dropdown active"><a href="#">About</a><b></b>
        <ul>
          <li><a href="#">Our History</a></li>
          <li class="active"><a href="#">Our Leadership</a></li>
          <li><a href="#">Our Community</a></li>
        </ul>
      </li>
      <li class="dropdown"><a href="#">Contact</a><b></b>
        <ul>
          <li><a href="#">Our Locations</a></li>
          <li><a href="#">Support</a></li>
        </ul>
      </li>
      <li class="last"><a href="#">Blog</a></li>
    </ul>
```

```
</nav>
</header>
...
```

This is a static menu with a toggle button that will be used to toggle the menu visibility at smaller device sizes. This menu does not currently represent the structure of the site. We have also included classes for active and first and last menu items, as well as classes for items that have child pages.

Next, add the following CSS to your `skin.css` file. The CSS is commented to walk you through the style for the menu. It includes media queries for smaller sized screens as well. You will not be putting these into the individual size specific stylesheets. However, you will move these to a menu-specific stylesheet a bit later.

```
/* Hides Toggle button on large screens */
#menuToggle{
    display:none;
}
/* Positions nav and displays as a table to vertically aligned
contents */
nav{
    display:table-cell;
    float:right;
    font-size:18px;
    font-weight:700;
}
/* Resets line-height from header and removes margins */
nav > ul {
    display:inline-block;
    line-height:1;
    margin:0;
}
/* Hides bullets */
nav ul li{
    list-style:none;
}
/* Sets position and sizes */
nav > ul > li{
    display:table-cell;
    padding:0 0 0 15px;
    position:relative;
    vertical-align:middle;
}
/* Adds vertical pipe and spacing around menu items */
nav > ul > li:after{
    content:'|';
    padding:0 0 0 15px;
}
}
```

```

/* Don't add pipe on last menu item */
nav > ul > li.last:after{
    content:none;
    padding:0;
}
nav ul > li > a:visited{
    color:#FFEDCB;
}
/* sets font styles */
nav ul li a{
    color:#FFEDCB;
    display:inline-block;
    text-decoration:none;
    text-transform:uppercase;
}
/* Hides caret on desktop */
nav ul > li > b {
    display:none;
}
/* Sets color for active menu items */
nav ul li.active > a{
    color:#D84D41;
}
/* Sets hover color */
nav ul li:hover > a,
nav ul li.active:hover > a{
    color:#3a83a7;
}

/* Menu dropdown */
/* Sets background color, positions dropdown,
   hides dropdown, and sets CSS transition */
nav > ul > li ul{
    background-color:#1F4B59;
    margin-left:-20px;
    padding-top:15px;
    max-height:0;
    min-width:120px;
    overflow:hidden;
    position:absolute;
    transition: all .5s;
    z-index:500;
}
/* Shows sub nav when hover over parent menu item */
nav ul li:hover ul{
    max-height:600px;
}
/* Removes padding and margins from child items */
nav > ul > li > ul li{
    margin:0;
    padding:0;
}

```

```

    }
    /* Sets padding and keeps text from wrapping */
    nav > ul > li > ul li a {
        padding:10px 20px;
        white-space:nowrap;
    }

/* Small Screens */
@media only screen and (max-width: 992px){
    /* Shows menu toggle button; sets width and styles button to be
round */
    #menuToggle {
        background-color:#15313F;
        border:1px solid #FFEDCB;
        border-radius: 35px;
        color:#FFEDCB;
        display: block;
        float:right;
        line-height:20px;
        margin:15px 15px 0 0;
        padding:15px 7px;
        text-decoration:none;
        text-transform:uppercase;
        transition: all .5s;
    }
    /* Shows hover style */
    #menuToggle.active{
        background-color:#FFEDCB;
        border:1px solid #15313F;
        color:#15313F;
    }
    /* sets background color */
    nav {
        background-color: #15313F;
    }
    /* Hides nav by default */
    nav,
    nav ul li ul {
        max-height: 0;
        overflow: hidden;
        transition: all 1s ease;
    }
    /* resets from desktop styles */
    nav,
    nav ul ul,
    nav > ul,
    nav ul li {
        clear: both;
        display: block;
        margin:0;
        padding:0;
    }

```



```

width: 100%;
}
/* shows nav after JavaScript adds open class to nav */
nav.open {
  max-height: 600px;
}
/* clearfix */
nav > ul > li:after {
  content: '';
  padding: 0;
  font-size: 0;
}
/* Adjusts item height to be better for touch */
nav > ul > li > a {
  line-height: 50px;
  padding: 0 15px;
}
/* Sets links to display as block to be better for touch */
nav ul li a {
  display: block;
}
/* Styles b element to be a toggle for sub nav */
nav > ul > li.dropdown > b {
  background-color: #1F4B59;
  border: 1px solid #FFEDCB;
  color: #FFEDCB;
  cursor: pointer;
  display: block;
  line-height: 25px;
  position: absolute;
  right: 10px;
  text-align: center;
  top: 10px;
  width: 25px;
  transition: all .5s;
  -webkit-transform: rotate(90deg);
  -moz-transform: rotate(90deg);
  -ms-transform: rotate(90deg);
  -o-transform: rotate(90deg);
  transform: rotate(90deg);
}
/* Rotates b element after JavaScript opens subnav */
nav > ul > li.dropdown.open > b {
  -webkit-transform: rotate(-90deg);
  -moz-transform: rotate(-90deg);
  -ms-transform: rotate(-90deg);
  -o-transform: rotate(-90deg);
  transform: rotate(-90deg);
}
/* Turns off Hover expanding for subnav */
nav ul li:hover ul {

```

```

        max-height:0;
    }
    /* Resets subnav from desktop view and hides */
    nav ul li ul {
        margin:0;
        max-height:0;
        position: relative;
        transition: max-height 1s;
        padding-top:0;
    }
    /* Shows subnav after Javascript adds open class to subnav */
    nav > ul li.open ul {
        max-height: 200px;
    }
}


```

Lastly, add the following JavaScript to the `scripts.js` file that you created in the Client Resource Manager section. This JavaScript will toggle the menu as well as the child menu items while on smaller devices.

```

$(document).ready(function () {
    $('#menuToggle').click(function () {
        $(this).toggleClass('active');
        $('nav').toggleClass('open');
        $('.dropdown').removeClass('open');
        return false;
    });
    $('.dropdown > b').click(function () {
        $(this).parent('li').toggleClass('open');
        return false;
    });
});

```

You now have a fully functional, yet static, responsive menu. Next, you will work at making it dynamic. To see the results of your work as you go, you first need to add some pages to test the navigation. While logged in as a site administrator or host, navigate to Admin  Page Management. Right-click the root of your site and select Add Page(s) from the drop-down. In the Pages text area to the right, add the following:

```

About
>Our History
>Our Leadership
>Our Community
Contact
>Our Locations
>Support
Blog

```

This will create the pages that you have in the sample HTML. Once completed with the DDR template, the menu should render the same as the static HTML example. When adding the new pages, they may not appear in the menu depending on the security settings of the pages. Make sure to go to each page's settings and set the permission rights to be viewable by all. Right-click the Contact page and select Disable Link in Navigation. We will explore the impacts of this a bit more later. Next, you will look at the pieces required to get the DDR Menu working.


DDR

The DDR Menu provider is more complicated than other skin objects, as it requires a template that represents the intended rendered HTML. We will cover some of the basic options in this example; however, there are many more configuration options available on the DNN Wiki that are not covered in this book (see <http://www.dnnsoftware.com/wiki/page/ddrmenu>). There are several files and attributes for the DDR provider that must be understood before creating the menu template.

Menu Style

The DDR Menu provider uses a menu style folder to define the design and layout of the menu. The menu style folder contains the template file, which can be token-, XSLT-, or Razor-based; a manifest file that determines the appropriate files to be used by the menu provider; and optional stylesheets and JavaScript files.

Menu File Structure

To create the menu style folder, right-click the `SubtleTrend` skin folder and select Add  New Folder. Name the folder `ResponsiveMenu`. This menu can be named whatever you choose; you will reference this folder in the `MenuStyle` attribute in the skin object for the skin.

DDR provides a lot of flexibility for building the template by providing options to create MVC Razor, XSLT, or token-based templates. While the Razor and XSLT approaches provide some great flexibility, the token-based approach will provide much of the functionality you will need on a normal basis. This approach will work perfectly for the menu in this example.

Menu Stylesheets

Right-click the `ResponsiveMenu` folder and select Add ➤ Add New Item. Select Style Sheet from the file types and name it `menuStyles.css`. This can be named whatever you choose, as it will be referenced later in the menu's manifest file. Move the CSS that you added for the menu in the `skin.css` file to this file and save it. You could leave the styles in the `skin.css` file and things would work perfectly fine; however, doing it this way will allow you to keep the styles for the menu together with the menu resources.

Menu JavaScript

Next, right-click the `ResponsiveMenu` folder and select Add ➤ Add New Item. Select JavaScript file from the file types and name it `menuScript.js`. Move the JavaScript from the `scripts.js` file that you created earlier to this file and save it. Again, moving it ensures that the JavaScript is loaded when the menu is.

Menu Template

The next step is to create the tokens template file. Right-click the `ResponsiveMenu` folder and select Add ➤ Add New Item. Select text file from the file types and name it `menuTokens.txt`. Move the HTML for the anchor for the Menu Toggle link as well as the `<nav>` element with all of its contents from the skin to this new file and save all files. We will come back to make the menu dynamic a bit later.

Menu Manifest

Lastly, right-click the `ResponsiveMenu` folder and select Add ➤ Add New Item. Select XML File from the file types and name it `menudef.xml`. This is the one file that must be named exactly as directed, as the menu provider references this file to assign the other files to their roles. Add the following content to this file and save it:

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <template>menuTokens.txt</template>
  <scripts>
    <script>menuScript.js</script>
  </scripts>
  <stylesheets>
    <stylesheet>menuStyles.css</stylesheet>
  </stylesheets>
</manifest>
```

This is the menu's manifest file, which declares which files to use for the templates, scripts, and stylesheets. This is where you reference the files that you just created. Next, you need to register and declare the menu provider in the skin.

Adding the Menu Provider to the Skin

Register the menu provider by placing the following code at the top of the file with the other control's register directives:

```
<%@ Register TagPrefix="ddr" TagName="MENU"
      Src="~/DesktopModules/DDRMenu/Menu.ascx" %>
```

Next, declare the skin object by replacing the contents of the header below the logo, as follows:

```
...
<header>
  <dnn:LOGO ID="siteLogo" runat="server" />
  <ddr:MENU ID="mainNav" MenuStyle="ResponsiveMenu" runat="server" />
</header>
...
```

You will see that the value for the `MenuStyle` is the name of the folder that contains the menu files. This is very important. The navigation provider will look for the manifest file in this folder in the root of the skin package. Save this file and refresh your site. You should see the menu as it was when it was in the skin file. If you do not, make sure you moved all of the code over to the appropriate files and are referencing the correct `MenuStyle` folder in your menu skin object in your skin.

There are several other options for the DDR Menu, such creating a certain HTML structure for the main menu and using a second instance of the menu provider with a different template for the footer where you may want to define an alternate HTML structure. The menu can also be configured to only show root menu items or only show the current or a specific page's children. Another great use of the DDR Menu is for use as a custom breadcrumb.

Menu Template

DDR Menu refers to pages as *nodes*. Use the `[*>NODE]` directive to define the location for the menu's template. Move all of the list items out of the root unordered list to another file to keep as a reference. Then, add the `[*>NODE]` directive inside the `ul` element, as shown:

```

<a href="#" id="menuToggle">Menu</a>
<nav>
  <ul>
    [*>NODE]
  </ul>
</nav>

```

This will cause the provider to repeat the HTML defined in the `[>NODE]...[/>]` template for each node (page) in this location. Below the closing of the `nav` element, add the following code:

```

...
</nav>
[>NODE]
  <li>
    <a href="#">[=TEXT]</a>
  </li>
[/?]

```

Refresh your page to see your menu. Well, some of it anyway. This displays the root menu only and the items still just link to `#`. You will define what happens with child nodes (pages) later. This template will repeat everything within the `[>NODE]...[/?]` directive for each item at the root level. The `[=TEXT]` token adds the text from the page name that is set in the page settings for each page. You will also notice that the last item still has the vertical pipe at the end, as defined by the CSS for each menu item, and the menu works on smaller screens but there are no toggles for submenus.

First, let's get rid of the vertical pipe from the last menu item. The CSS references the class of `last` to get rid of the vertical pipe, so you want to make sure that the last list item has this class. The DDR Menu includes some very handy conditional directives that allow you to test for certain situations and render different HTML based on those conditions. If the condition is true, anything within the directive will be displayed. So, you will use the `[?LAST]...[/?]` conditional to test whether it is the last node. For good measure, you will also use the same technique to add the `first` class to the first list item.

Update your `[>NODE]` template as follows:

```

[>NODE]
  <li class="[?FIRST]first [/?][?LAST]last[/?]">
    <a href="#">[=TEXT]</a>
  </li>
[/?]

```

Save the file and refresh your page. Using your browser's developer tools to

inspect the menu items, you will see that the first list item has the class of `first` and the last list item has the class of `last`. You will also see that the vertical pipe on the last list item is now gone.

While you are adding some helpful classes, you still need a couple more to be added to the list items. Using the `[?NODE]...[/?]` conditional, you can test to see if the page has children and apply the `dropdown` class if the condition is true. Using the `[?SELECTED]...[/?]` conditional, you can test to see which node is the current page and apply the `active` class to that list item:

```
[>NODE]
  <li class=" [?NODE]dropdown [/?][?SELECTED]active [/?]
    [?FIRST]first [/?][?LAST]last [/?]">
    <a href="#">[=TEXT]</a>
  </li>
[/?]
```

Refresh your browser and inspect the element with your browser's developer tools. You will see that the pages that have children pages now have the class of `dropdown` on the list items. Also, the current page is highlighted in orange because it now has the class of `active` applied to the list item for that node. Now, let's add a few more tokens to build the rest of the navigation. Add the `[=URL]` token to the `href` attribute of the anchor as well as the `target` attribute with a value of `[=TARGET]`. This will allow the page to open to a new browser if it has been set to open into a new page in the page's settings. Using the `[?NODE]...[/?]` conditional again, you can add the chevron to the nodes with children for viewing at a smaller resolution by adding the `b` element. Your `[>NODE]` template should now look like this:

```
[>NODE]
  <li class=" [?NODE]dropdown [/?][?SELECTED]active [/?]
    [?FIRST]first [/?][?LAST]last [/?]">
    <a href=" [=URL] " [?TARGET] target=" [=TARGET] " [/?]">[=TEXT]</a>
    [?NODE]
    <b>></b>
  [/?]
</li>
[/?]
```

Save your file and refresh your page. You can now navigate through the root page level of your site. You will also notice that, when viewing at smaller resolutions, the chevrons now appear for your parent pages. Things are looking great right now, but you have just a couple of more things to take care of. DNN includes the ability to disable a page as you did for the Contact page

after you created the test pages. Clicking the menu item for that page will produce a 404 message, letting you know that the page could not be found. This is obviously not an experience you want for your users. DDR Menu has another conditional that will work perfectly for this. You can create two versions of the link, one with a URL to navigate with and another with an # as a placeholder. You can also use a span instead of an anchor, but for this example, we will stick with the link. Update your `[>NODE]` template as follows:

```
[>NODE]
  <li class="[?NODE]dropdown [/?][?SELECTED]active [/?]
    [?FIRST]first [/?][?LAST]last[/?]">
    [?ENABLED]
      <a href="[=URL]" [?TARGET] target="[=TARGET]" [/?]>[=TEXT]</a>
    [?ELSE]
      <a href="#" class="disabled">[=TEXT]</a>
    [/?]
  [?NODE]
    <b>></b>
  [/?]
</li>
[/?]
```

Notice the `[?ELSE]` conditional directive. It allows you to check to see if a condition is true or false and then provide a different pattern depending on the results returned. For the disabled page, we have removed the `[=URL]` token, no longer have a `target` attribute available, and have added a class of `disabled`.

Lastly, you need to add your child pages. Using the `[?NODE]...[/?]` conditional directive that you created to add the chevron, you can add the directive to build out the child pages with the same template. You do this the same way that you specified the root menu in the `nav` element. This will direct the menu provider to use the same `[>NODE]...[/?]` template for all child pages as well. Now, your entire template should be as follows:

```
<a href="#" id="menuToggle">Menu</a>
<nav>
  <ul>
    [*>NODE]
  </ul>
</nav>
[>NODE]
  <li class="[?NODE]dropdown [/?][?SELECTED]active [/?]
    [?FIRST]first [/?][?LAST]last [/?]">
    [?ENABLED]
      <a href="[=URL]" [?TARGET] target="[=TARGET]" [/?]>[=TEXT]</a>
```



```
[?ELSE]
  <a href="#" class="disabled">[=TEXT]</a>
[/?]
[?NODE]
  <b>></b>
  <ul>
    [*>NODE]
  </ul>
[/?]
</li>
[/>]
```

Refresh your page and the sub-navigation will now show when the parent page is hovered. Resize your browser to a smaller resolution; notice that the chevron button has been added to the menu items that contain child pages. Clicking this button will toggle the sub-navigation.

Creating Alternate Skins

The skin is pretty flexible for allowing different layout options, but what if you want to have a banner and a three-column layout? What if you only want to have a full-width layout, which is great for admin pages? You can create other skins in the same skin package and reuse the styles.

You will create a second, full-width skin that will be great for applying to admin pages and any other pages that might require the page's full width. You can repurpose the `home.ascx` skin by creating a copy and making a few adjustments. Copy and paste your `home.ascx` skin in your

`~/Portals/_default/Skins/SubtleTrend` folder and rename it `full-width.ascx`. Navigate to the About page and change the page settings to use the new `Host: SubtleTrend - full-width` skin. Your page should look the same as your other pages with your `home.ascx` skin, as it is just a copy right now. Add a new HTML module to the Content Pane and add some test content to the module.

Edit the `full-width.ascx` skin and remove the `TopPane` div and the entire `contentWrap` div, including its contents. Change the `LowerPane` ID to `ContentPane` and change the class to `content-pane`.

If you refresh your page now, you will see that your skin stretches the full width of the browser and is not centered to be consistent with the rest of your design. You can very easily fix this with a skin-specific stylesheet. Right-click the `SubtleTrend` skin folder and select Add → Add New Item. Select Style Sheet from the file types, name the file the same name as the skin (`full-width.css`), and click the Add button. Replace the contents with the following CSS and save the file:

```
.content-pane{
    float:none;
    margin:45px auto;
    width:1170px;
}
@media screen and (min-width:992px) and (max-width:1199px){
    .content-pane {
        width: 960px;
    }
}
@media screen and (max-width:991px){
    .content-pane {
        width: 100%;
    }
}
```


}

You now have a full width version of your skin. Refresh your page and you will see that it is now full width.


Creating Containers

Containers resemble skins by providing a specific look and feel; however, they are specified at a module level instead of the page level. They can be used to apply different styles to different modules. If you note in the design, the title in the main section is orange, and the background is the same color as the page. The content in the Aside Pane, however, has a beige title and a darker blue background. By contrast, the lower content section has a beige background with blue text. Not to mention the top banner section has the title in a large circle with other content to the right of the page. One approach to achieve this design would be to set these styles based on the content panes, and in some cases that is preferable, as it can keep the design a bit more consistent. However, creating flexible styles that can be applied at the module level will allow you to move these styles around on the page.

Creating the Folder Structure

The process of creating a container is similar to the process of creating a skin. They follow the same rules for site permissions. If you want your containers to be available to the entire DNN installation, you add your container package to the `~/Portals/_default/containers` folder. To make the containers be available for your specific site, you need to create a containers folder within your specific site folder, such as `~/Portals/0`, where 0 is the ID of your site. As with the skin, you will create the containers in the `_default` folder to allow them to be available at the installation level. Right-click the `~/Portals/_default/containers` folder, select Add  New Folder, and name it the same as the skin package, `SubtleTrend`.

Creating Your First Container

You'll start by creating the container that is used in the Aside Pane in the design. It has a beige `h2` title and a dark blue background. Right-click the `SubtleTrend` folder and select Add  Add New Item. Select Web User Control from the C# section. Uncheck the Place Code in Separate File option and give it a name of `H2 Title - Blue.ascx`. Click the Add button. Make sure to name your container something that makes it easy to identify when adding to the page later. Replace all of the code in the file with the following code and save the file:

```
<%@ Control Language="C#" AutoEventWireup="false" Explicit="True"
```

```

    Inherits="DotNetNuke.UI.Containers.Container" %>
<%@ Register TagPrefix="dnn" TagName="TITLE"
Src="~/Admin/Containers/Title.ascx" %>

<section class="dnn_container h2-title container-blue">
  <h2><dnn:TITLE runat="server" id="dnnTITLE" CssClass="title" />
</h2>
  <div id="ContentPane" runat="server" />
</section>

```

Let's review what you just did. As with the skin, you first declare that this is a user control. The difference with this definition is that instead of inheriting from the `DotNetNuke.UI.Skins.Skin` class, you inherit from the `DotNetNuke.UI.Containers.Container` class. Next, you see that you registered a skin object for the module's title. These are similar to the skin objects for skins; however, these are available only within containers. Below that, you create a `section` element. This is the perfect place to use HTML5's `section` element as each module that is placed on the page will be its own new section of content and will comply with the HTML5 recommendations. You have also added a few classes to the `section` to allow for some flexible styling with the CSS.

Next, you add the container `TITLE` skin object inside an `h2` element and gave it a class of `title`. This will render the following HTML:

```

<h2>
  <span id="dnn_ctr415_dnnTITLE_titleLabel" class="title">
    Serif on the <aside>
  </span>
</h2>

```

Note that DNN added a `span` inside the `h2` element. This is something that can create issues when styling your headers, so be sure to include style definitions for both `h2` and `h2 span`. Also note that you created a new content pane within the container with the `id` assigned the value `ContentPane`. Similar to skins, containers must have a content pane and it must be named `ContentPane`; however, each container can have only one content pane.

Once you have created your new container, log in as an administrator or host and hover over the module settings icon for the module in the Aside Pane (Serif on the `<aside>`). Select the Settings option from the drop-down as you did when setting the module's title. On the Page Settings tab, in the Basic Settings section, select `Host: SubtleTrend - H2 Title - Blue` from the Module Container options. Scroll to the bottom of the window and click the

Update button. You may not notice much of a difference, if any, but once you add some CSS, you will see the beauty of your new container.

Additional Skin Objects for Containers

There are a few other skin objects for containers, but (other than the ICON skin object) they are rarely used. The ICON skin object can be useful for applying icons or module header images. To use the ICON skin object, first register it in the container as follows:

```
<%@ Register TagPrefix="dnn" TagName="ICON"
Src="~/admin/Containers/icon.ascx" %>
```

Then, add the skin object to the desired location in your container's HTML with the following code:

```
<dnn:ICON ID="containerICON" CssClass="container-icon" runat="server"
/>
```

The image can be selected under the Page Settings tab in the Module Settings dialog in the Basic Settings section labeled “Icon.” For example, if you set the icon to use an image file in the root of the site folder, with the name `myImage.png`, the following HTML would be rendered to display the image:

```

```

The skin object gives the icon an `id` that is specific to the module that it is in. It also uses the title of the module as the `alt` tag for the image. If no image is specified in the module's settings, the skin object will not be rendered to the page, leaving your icon optional.

Styling Containers

Another similarity of the containers to the skins is the automatic detection of stylesheets. A stylesheet named `container.css` in the container's folder will apply when any container from the container package has been added to the page, and a stylesheet with the name of a specific container will be added when that specific container is added to the page. This brings up a point to be aware of—containers from other container packages can contain styles that will conflict with the styles you have created for your containers. Because of this, it is a good idea to create all of the containers that you will need and

make sure not to include any containers from other container packages.

Although styling containers is similar to styling skins, you are going to take a slightly different approach for this skin. Instead of creating a `container.css` file in the container's folder, you will be adding styles to the `skin.css` file. This will keep the CSS in one place, thereby making it easier to find and edit at a later time. This will also be one less resource that is needed to load on the page through the form of another CSS file if you are not consolidating your files with the Client Resource Manager. You also need to include some styles for your responsive designs to the size-specific stylesheets.

At the bottom of the `skin.css` file, add the following style definitions:

```
/** Container Styles **/  
  
/* Blue Container */  
/* Set container background and add padding */  
.container-blue{  
    background-color:#15313F;  
    padding:45px;  
}
```

Refresh your page to see the new style added to the container in the Aside Pane.

Creating Additional Containers

Next, you will create several new containers. Some may not be used in this design, but you will see how they will provide nice flexibility in the page layout. Let's create a similar container but without a title. Copy the new container that you created earlier (`H2 Title - Blue.ascx`) and paste a duplicate in the containers folder. Rename the duplicate `No Title - Blue.ascx`. Open the file and remove the `<%= Register ... %>` declaration for the title and the `h2` element with the `Title` skin object. Change the class `h2-title` on the `section` element to `no-title`. The code for your container should look like this:

```
<%@ Control Language="C#" AutoEventWireup="false" Explicit="True"  
    Inherits="DotNetNuke.UI.Containers.Container" %>  
<section class="dnn_container no-title container-blue">  
    <div id="ContentPane" runat="server"></div>  
</section>
```

Duplicate the `H2 Title - Blue.ascx` and `No Title - Blue.ascx` files and rename them `H2 Title - Beige.ascx` and `No Title - Beige.ascx`,

respectively. Edit the class for the section element in both containers from `container-blue` to `container-beige`. Then, update the module in the lower pane to use the `Host: SubtleTrend - No Title - Beige` container in the module settings dialog.

Add the following styles to the bottom of the `skin.css` file:

```
/* Beige Container */
/* Set container background and padding */
.container-beige{
    background-color:#FFEDCB;
    padding:30px;
}
/* Specify color of header and text in body of container */
.container-beige h2,
.container-beige .Normal{
    color:#15323F;
}
```

Once more, duplicate both containers and name them `H2 Title.ascx` and `No Title.ascx`, respectively. Edit both containers and remove the `container-blue` class from the section element. This will provide a default, basic container with no extra color variations.

Next, you'll create a container with an `h1` for the title. Duplicate the `H2 Title.ascx` container and rename it `H1 Title.ascx`. Edit the container and change the class of `h2-title` to `h1-title` and change the `h2` element to an `h1` element.

Add the following CSS to your `skin.css` file:

```
/* H1 Container */
/* Specify styles for H1 with colors, font sizes and spacing */
h1 .title{
    color:#D84D41;
    font-size:150px;
    font-weight:900;
    line-height:.8;
    text-transform:uppercase;
}
/* Change color of nested spans to orange */
h1 .title span{
    color:#FFEDCB;
}
```

Recall that you named the module in the content pane `Lorem
 &#amp; Ipsum`. Update this module to use the new `H1 Title`

container. DNN allows you to use HTML on your module titles. This is very handy when you want to create some visual styling that can be reused. Anytime you want to use the beige as a secondary color on the `h1`, you just need to wrap the text you want to accent in a `span` element and it will pick up this style.

Lastly, you are going to create the banner container. This can typically be set up with the simple “No Title” container and an HTML module, but you are going to learn how to use containers to create some advanced styling that can be reused. Duplicate the `H1` container and rename it `Banner - Circle.ascx`. Edit the file and rename the class `h1-title` on the section element to `banner-circle`. Surround the `h1` and content pane with a simple `div` element that you will use for centering the content on the page. Update the module in the Top Pane to use the newly created `Host: SubtleTrend - Banner - Circle` container.

Then, add the following CSS to the `skin.css`:

```
/* banner */
.banner-circle{
  background-color:#15313F;
  padding:10px 0;
}
.banner-circle > div {
  margin:0 auto;
  width:1170px;
}

.banner-circle h1{
  display:table-cell;
}
.banner-circle h1 span{
  background-color:rgb(255,238,204);
  border-radius: 360px;
  color:rgb(21,50,63);
  display:table-cell;
  font-family:"Arial Black", Gadget, sans-serif;
  font-size:80px;
  font-weight:900;
  height:360px;
  letter-spacing:-7pt;
  line-height:75%;
  padding-left:12px;
  text-align:left;
  vertical-align:middle;
  width:360px;
}
```

```

.banner-circle > div div {
  display:table-cell;
  vertical-align:bottom;
  width:810px;
}

.banner-circle .Normal p{
  color:#1f4b59;
  float:right;
  font-family:Georgia, "Times New Roman", Times, serif;
  font-size:72px;
  line-height:83%;
  text-align:right;
  text-transform:uppercase;
  vertical-align:middle;
  width:400px;
}

.banner-circle .Normal p strong {
  font-size:90px;
  font-weight:bold;
  line-height:80%;
}

```

This has a lot of custom CSS3 and layout, but you can see that you took a simple `Text` module and created a complex design that can be reused on any page using basic content.

To make these container styles responsive, add the following styles to their respective stylesheets:`Media-medium-screens.css`

```

/* Reduce size of H1 */
h1 .title{
  font-size:90px;
  line-height:.8;
}

/* Banner */
/* Reduce width of banner container */
.banner-circle > div {
  width:960px;
}
/* Reduce size of Circle */
.banner-circle h1 span{
  font-size:53px;
  height:240px;
  letter-spacing:-5pt;
  padding-left:12px;
  width:240px;
}
/* Reduce size of right side of banner */

```

```
.banner-circle > div div {
  width:722px;
}
/* Reduce Subtle text in banner (HTML content) */
.banner-circle .Normal p{
  font-size:52px;
  line-height:83%;
  width:400px;
}
/* Reduce strong text in banner (HTML content) */
.banner-circle .Normal p strong {
  font-size:60px;
  line-height:80%;
}
```

Media-small-screens.css

```
/* Reduce size of H1 */
h1 .title{
  font-size:70px;
  line-height:.6;
}

/* Banner */
/* Reduce width of banner container */
.banner-circle > div {
  position:relative;
  width:100%;
}

/* Reduce size of right side of banner */
.banner-circle > div div {
  display: block;
  position: absolute;
  right: 15px;
  top: 15px;
  width: 322px;
}
.banner-circle h1 {
  padding-left:30px;
}
/* Reduce size of Circle */
.banner-circle h1 span{
  font-size:41px;
  height:180px;
  letter-spacing:-4pt;
  padding-left:7px;
  width:175px;
}
```

Media-xtra-small-screens.css

```

/* Reduce size of H1 */
h1 .title {
    font-size: 45px;
}
/* Reduce size of H2 */
h2 .title {
    font-size: 35px;
}

/* Banner */
/* Make banner container fluid width */
.banner-circle > div {
    width: 100%;
}
/* Center circle and reduce size */
.banner-circle h1 {
    display: block;
    padding-left: 0;
    margin: 0 auto;
    position: relative;
    width: 175px;
}

/* reduce size of banner text */
.banner-circle h1 span {
    font-size: 41px;
    letter-spacing: -4pt;
    padding-left: 7px;
}
/* Move module content below title and make fluid */
.banner-circle > div div {
    display: block;
    position: relative;
    right: auto;
    top: auto;
    width: 100%;
}
/* Center module content */
.banner-circle .Normal p {
    float: none;
    margin: 0 auto;
    padding: 30px 0;
    text-align: center;
    width: 290px;
}

```


You have now completed your first skin! Refresh and then resize your browser to see how nice your skin looks on all resolutions and sizes!

Custom 404 and Pop-up Skins


DNN includes special skins for pop-ups and 404 pages. There are some considerations for these skins that should be noted. The 404 page requires that there aren't any links that create postbacks on your page. This means that you must create a skin that does not include any postbacks, such as the register and login skin objects.

A customized pop-up skin can be added by creating a new skin with the name `popUpSkin.ascx`. To allow DNN to recognize your pop-up skin, you must first set the Site Skin on the Admin [↗](#) Site Settings page in the Basic Settings [↗](#) Appearance section to a skin within your skin package. Add the basic code required for a skin (see the “ASCX Approach” section in the “Creating Your First Skin” section of this chapter) to your skin file. This will cause the pop-up to inherit the `skin.css` file. You can also create a specific CSS file for your pop-up skin by creating a new stylesheet called `popUpSkin.css`. Styles in this file will override any styles specified in the `skin.css` file. This skin will only apply to the contents of the Iframe of the pop-up window and could still require some edits to DNN's pop-up classes.

Skin Thumbnails

It is best to create skin preview images for each of your skins and containers to assist content editors when they're selecting a skin to display their content. Navigate to Admin  Skins and select Gravity as the option for the Skins selection in the Skin Editor section. Notice that most of the skins have a thumbnail view of each of the skins and containers. Click the thumbnail of one of the skins and you will see that it opens a new view with a sample screenshot of the skin. Switch the Skins option in the Skin Editor to the `SubtleTrend` skin and you will notice that it displays an "Image Not Available" placeholder.

If you were provided a design for each of your skins, you can use a JPG version for each one. Otherwise, you should create a page for each skin and add some content to the pages. Then, take a screenshot of your page. Once you have a screenshot or an exported design file, it is a good idea to add some borders on your images to designate each content pane. This will be very useful to the content editors. Once you have designated the content panes, save each file as the name of the skin it represents and place it in the skin file. For example, you could create a `Home.jpg` for the `Home.ascx` file and a `full-width.jpg` for the `full-width.ascx` file.

Next, create sample screenshots of the containers. It is helpful to use the same content and switch out the containers to show what the differences are for each container. These images will be displayed when clicking the thumbnails of each skin; however, you need to create the thumbnails for each image. This is the easy part. Refresh the Admin  Skins file with the `SubtleTrend` skin selected. Done. DNN just created the thumbnails for your skins.


Creating an Installable Skin Package


So far you have created a skin package with two skins. This skin package can be used on other DNN installations; however, the files must be copied directly to the `skins` and `Container` folders for the other installations. This is not always feasible depending on your access to those environments. It could be that you are creating skins that you want to sell and the people buying them have only administrative knowledge of DNN. It would be a good idea to have a skin package that can be installed just as easily as modules are. DNN provides a process for doing this in an installable skin package.

Packaging the Skin

Creating a skin package requires a few steps. First you have to create a new extension for the skin and then create the skin package for that extension. Next, you repeat that process for the container, creating an extension and package. Lastly, you merge the two packages together to create one installable package for both the skins and containers.

Creating the Skin Package

Creating a skin package can be handled through the DNN interface. It is best to perform this step after you have completed all development and testing of your skin, as updating the installation package isn't a small feat. Before you begin the process to create the skin package, it's best to create the License and Release notes files. Right-click the `SubtleTrend` skin folder and select Add  Add New Item. Select the Text File file type and name it `license.txt`. Add your license information for your skin in this file. Repeat the process for the release notes by creating a `release.txt` file and adding your release notes.

Next, you can begin by creating the skin extension. While logged in as a host, navigate to the Host  Extensions page. Click the Create New Extension button on the top-right of the page, above the list of modules. Select Skin as the extension type. Add a name (can't have spaces and must be the same name as the skin folder for DNN to recognize there is a matching skin package), a friendly name, a description, and a version number (v 01.00.00) for your skin. Click the Next button.

Enter the owner details and click the Next button. Your extension has been created, but your skin package has not. Right now, it is basically a shell for your skin package that needs to be exported. Scroll down to the Skins section


and click the Edit icon for your new skin extension.

Leave the License and Release Notes fields empty, as you already created those files. We will look at how to include these external files in a later step. Click the Create Package button. Review the information for your skin package and click the Next button.

In the Choose Files to Include section, review the files to be included and remove any that you don't want to include in the package. Click the Next button. Next, on the Create Manifest section, update the `license` and `releaseNotes` tags to reference the following, and click the Next button:

```
<license src="license.txt" />
<releaseNotes src="release.txt" />
```

Confirm the information in the Create Package screen and click the Next button. Your skin package has been created and can be found in the `~/Install/Skin` folder of your site. Find that folder and you should see your skin package in there.

Repeat the process for the containers by navigating to the Host  Extensions page. Click the Create New Extension button. Select Container as the extension type. Because you have already created an extension with the name `SubtleTrend`, you will not be able to create it with the same name. Instead, name it `SubtleTrendContainer`. Complete the rest of the information and click the Next button. Feel free to skip the information for the owner details section if you plan to merge with the skin package, as it will already contain all of this information. Click the Next button.

Since you were not able to create the extension with the name and the extension name must match the container package folder, you must rename it. Expand the Containers section on the Extensions page and edit the `SubtleTrendContainer` package by clicking the Edit icon. Edit the Skin Package Name by removing “Container” from the name, which changes it to `SubtleTrend`. Click the Update Extension button. Close the pop-up by clicking the Cancel button or clicking the X in the top-right corner. Edit the container extension once again by clicking the Edit icon. This time you will see a Create Package button. Click that button. Review the information on the Create Package section and click the Next button. Review the files in the Choose Files to Include section and click the Next button. Review the manifest created in the Create Manifest section and click Next. Review the information in the Create Package section and click the Next button. Once the package is

successfully created, click the Return button at the bottom of the pop-up window.

You now have two installable packages, a skin package and a containers package. While you can install each individually, it is best practice to deliver the skin and container as one package.

Unzip the install package for the skins and edit the manifest file (the file with the name of your skin package and a `.dnn` file extension). Open the manifest file for the containers and copy the contents (`<component type="Container">...</component>`) in the section `<components>` tag and paste it in the `<components>` section after the `<component type="Skin">...</component>` section. Save the file. Copy the contents of the ZIP file for the containers install file, except the manifest file, and paste them into the unzipped skins folder. From within the skins folder, select all files and zip them. You now have a fully installable skin and containers package.

Understanding the DNN Manifest

DNN uses an XML file with the `.dnn` extension when installing a skin package. Let's dissect this file a bit. The first section is the `<package>` section, which gives the name, type, and version of your skin. When updating your skin to a new version, the version attribute can be incremented to reflect the skin package's version number. The next few sections provide the friendly name, the description, and a link to an optional icon file. The following `<owner>` section provides information about the owner, which is helpful for guiding the user to know who to contact for support of the skin. The next two sections are the `<license>` and `<releaseNotes>` sections. The license and release notes can be either inline or, as in this example, links to external files. These files can contain HTML so that the information is presented in a friendly way during installation.

The next section is `<components>`. This contains all extensions that will be installed. This example has two extensions together as one package. This is where you add a reference to another extension if you want it to be installed along with your skin. This is very helpful if your skin is dependent on a third-party skin object.

Each component includes the name for each extension as well as the target location where the extension will be installed. In this example, using the container, it will be made available to the entire DNN installation. That's

because it's added to the `~/Portals/_default/Containers` folder as directed by the following code:

```
<basePath>Portals\_default\Containers\SubtleTrend</basePath>
```

Following the base path are the files that are included as a part of the package. These include a name of the file and a `<path>` if not located in the root folder. Upon examining the manifest file for the skin, you can see that all of the files for the menu have a path of `ResponsiveMenu`, which is the folder in which they are located.

Advanced Skinning Techniques

This section covers some additional techniques that can be helpful for creating some more customization to your skins.

Site Title for Logo

For the example you have been using for this site, the only image that you have is the logo. However, it might be a requirement that the logo be text and apply some font styling with CSS. You can use the title of the site to keep this dynamic. For the `SubtleTrend` skin example, use the following code:

```
<a href="/" class="logo"
  title="<%= PortalSettings.PortalName %>">
  <%= PortalSettings.PortalName %>
</a>
```

This will add a link with the site title being applied as the value for the `title` attribute of the link as well as the text in the link. You will also need to make some modifications to the CSS to make this work in the `SubtleTrend` design.

Page ID for Page-Specific CSS

Another useful technique is when you need to apply some CSS to a specific page, but it would be overkill to create a new skin. You can wrap the entire skin in a `div` and apply an `id` using the page ID with the following code:

```
<div id="pageId_<%= PortalSettings.ActiveTab.TabID %>">
  <header>...</header>
  ...
  <footer>...</footer>
</div>
```

If the page with this skin applied to it has a `tabId` of 84, it would define the `id` on the `div` of `pageId_84`. The `pageId_` is prepended to ensure that it will be a unique ID on this page. This allows you to reference the `header` of this specific page in your CSS by using the following:

```
#pageId_84 header {...}
```

Or you can use the page name instead of the `tabID`:

```
<div id="page_<%= PortalSettings.ActiveTab.TabName.Replace(" ", "")
%>">
  <header>...</header>
```

```
...
  <footer>...</footer>
</div>
```

If you were to apply this to the skin on the Our History page, it would produce the `page_OurHistory` ID, thereby allowing you to reference elements on this specific page. The `.Replace(" ", "")` removes any spaces from the page title. To reference the header on this page, you would use the following CSS:

```
#page_OurHistory header {...}
```

Page Icon as Header Image

Another handy technique is to use the built-in Page Icon setting to apply an image to a page header. Open the page settings and apply an image to the large page icon. Then add the following code to your skin where you would like the image to appear:

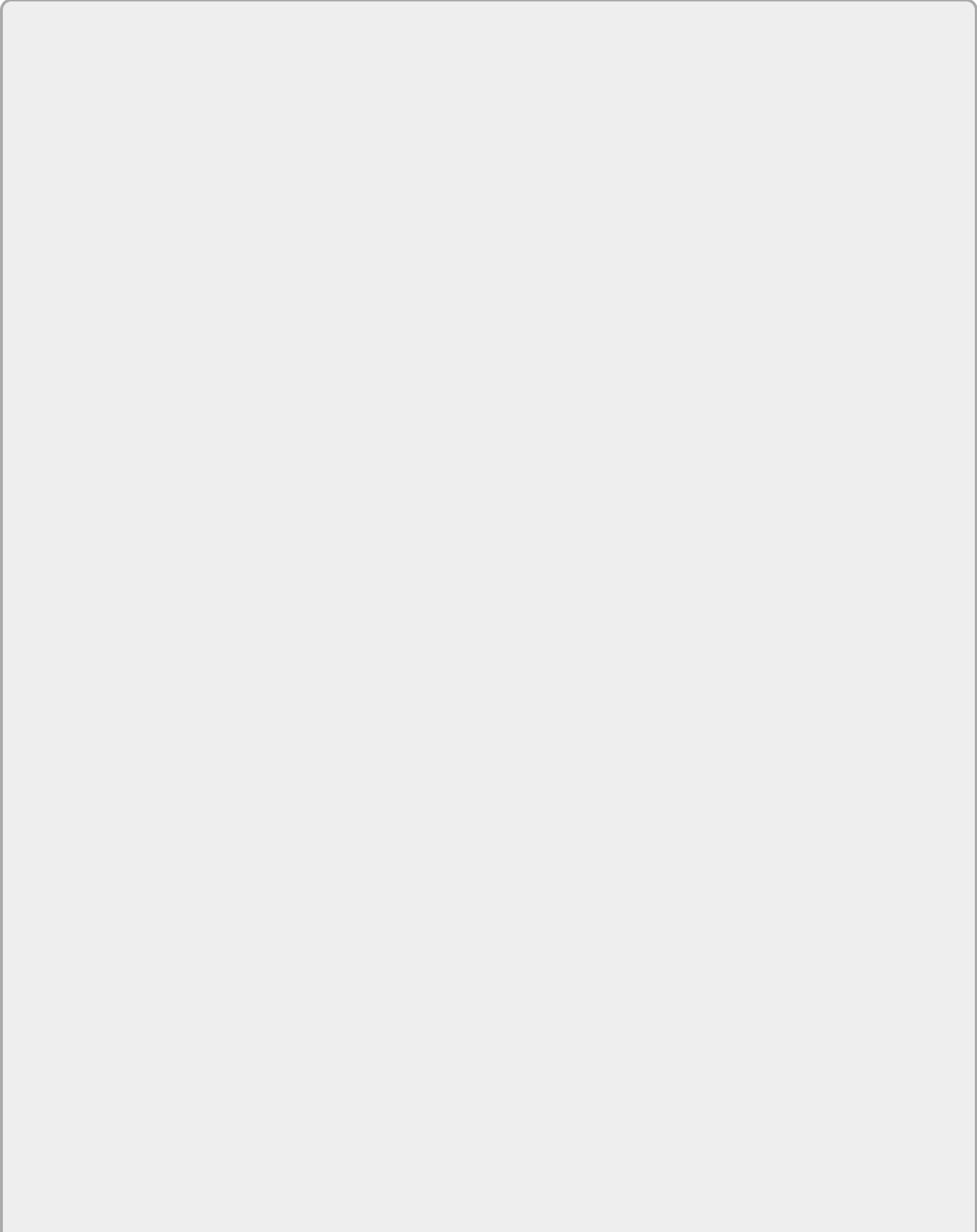
```

```

Summary

This chapter provided a look into understanding all of the integral parts of creating a DNN skin package. You learned what skin, container, skin object, and skin package are. You learned how to create dynamic navigation with a lot of flexibility to match your design. You learned how to leverage the built-in tools in DNN to make your site responsive to various display sizes so that your site is usable on mobile devices. Skinning is a constantly changing area of DNN development, so be sure to visit DNN Software's website and the community blogs. Also check out the author's site at <http://www.ralphwilliams.com> for some helpful skinning tips and tricks.

Chapter 18
Packaging and Distribution



What You Will Learn In This Chapter

- Creating a new extension
- Creating extension packages
- Editing the manifest file
- Controlling how an extension is installed
- Tailoring your package to exact specifications

DNN's commercial and free module market has been instrumental in the success of the platform. The ability to do a quick search on the Internet and find free or relatively inexpensive modules that fulfill a need significantly reduces the cost to operate a website when compared to a custom website and other commercially available platforms. With the wide potential customer base, it's no wonder that developers have filled the void with low- to medium-cost solutions to countless business needs. This market is no accident. The modular nature of DNN and the extensibility and distribution model have created an environment from which developers can easily share the modules they create and designers can easily share the skins they design. This chapter explains the extensions model for packaging and distributing modules, skins, language packs, and all other application extensions available to DNN developers and designers.

The New Extensions Model

In previous versions of DNN, installing modules, skins, languages, and other extensions was achieved using a different process for each type of extension. Skins were packaged using convention over configuration, meaning that if you put the correct files with the correct filename in the correct structure, it would install correctly. Modules were installed through the Module Definitions interface using a DNN manifest file that instructed the installer where and how to install modules. In version 4.6, a new installer was introduced to handle authentication system installations. All of these different models created more code to maintain and was an inconsistent experience for developers who had to learn a different way to package each type of extension they wanted to distribute.

In version 5, DNN consolidated all of these installation models into a unified model called *extensions*. This new model provides developers and administrators a single place through which all add-ons to the core are managed. By using a common interface, module developers can reuse the same skills for distributing modules, providers, authentication systems, skins, and even custom extension types. The new model also adds license agreements, release notes, and an uninstall process—features that were previously unavailable for skins and language packs.

Extensions can be managed by host- and administrator-level users to varying degrees. The Extensions page under the Admin menu for a site allows site administrators to control which modules are available to users with edit permissions based upon role. With this feature added in DNN 5, administrators have more granular control over the types of content that page editors can add to pages they have access to edit. For more information on managing the modules available to users with edit permissions, see [Chapter 4](#), “Site Administration.” Authentication systems and skins can also be managed from the Admin extensions page. The extensions page under the Host menu gives complete control to host users to manage authentication systems, containers, language packs, libraries, modules, providers, skins, and skin objects. The Host extensions page is also where the interface for installing and uninstalling extensions is located.

Creating New Extensions

To add custom extensions to your installation, you need to begin by making DNN aware of the extension. For developers, this is generally one of the first steps taken to start developing and testing modules and providers once the file and folder structure is ready. Although it is possible to create a package configuration file from scratch and build the extension package to install into your site, the simpler way in most cases to build your extension package is through the extensions management interface from the Host menu. After you create your package, you can manage the controls and files for the package and even export an installable extension package using the wizard. This section explains how to use the extensions management interface to create and distribute extension packages.

To begin the process of creating your extension, log in with your Host account and navigate to the Extensions page in the Host menu. As an example, you will create an extension for the `SkinningDemo` skin from [Chapter 17](#). Begin by clicking the Create New Extension button at the top of the page. The first screen in the New Extension Wizard gathers basic information about your extension. For the example skin package, change the extension type to `Skin`, set the name to `SkinningDemo`, the Friendly Name to `Skinning Demo Skin`, enter a short description, and set the version to `1.0.0`. When your inputs match [Figure 18.1](#), click Next to go to the next step in the wizard.

DNN 7 > Extensions > New Extension Wizard

Create New Extension

The Create New Extension Wizard allows you to create new Extensions. Enter the information requested in each step. Please select the type of Extension to create.

Select Extension Type:

In the first step please provide a unique name for the new Extension, - it is recommended that you use the format CompanyName.Name to avoid potential clashes if you intend to distribute the extension. In addition, you should provide a friendly name for the Extension, a description for the Extension and you should select a version (or the default version No. 0.0.0 will be used)

Name:

Friendly Name:

Description:

Version:

[Next](#) [Return](#)

[Figure 18.1](#)

The Owner Details screen is the last step in the New Extension Wizard for skins. [Figure 18.2](#) shows a typical Owner Details screen content. Once your input matches [Figure 18.2](#), click Next to add your extension to the DNN installation.

DNN 7 > Extensions > New Extension Wizard

Owner Details

In this last step you can optional provide details about the owner of the Extension. This information is very useful if you plan on distributing your extension.

Owner:

Organization:

Email Address:

[Next](#) [Return](#)

Figure 18.2

Now that DNN knows about your extension, you can edit the details of your extension from the Host extensions page. Find the Skins section in the Installed Extensions list (you may want to collapse any open sections). Click the pencil next to the `Skinning Demo` skin to enter edit mode for the extension. This common page allows you to manage the details of your extension. Regardless of the type of extension you are creating, there are common attributes associated with all package types that are managed in the Package Settings section on the details page for an extension. [Table 18.1](#) explains each of the attributes in the Package Settings section.

Table 18.1 Package Settings Attributes

Setting	Purpose
Name	This name must be unique because it is used in site and page templates to refer to modules and extensions.
Type	Setting that instructs DNN whether to treat the extension as a module, skin, provider, or any other custom extension type.
Friendly Name	The setting for display to end users. It is used in drop-downs, lists, and the control panel for modules.
Description	Setting to display a short description of the extension in the list of extensions on the Host and Admin extensions pages.

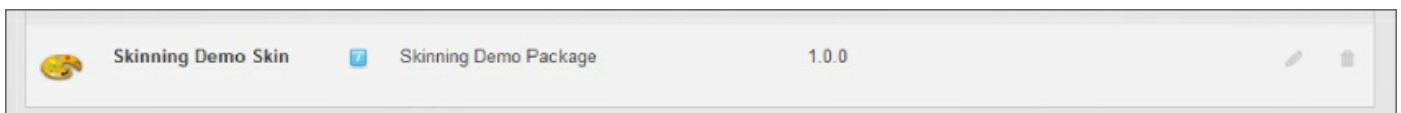
Version	The version of the extension.
License	A description of the terms of use for the extension.
Release Notes	Notes about the current version of the extension.
Owner	The owner of the package.
Organization	The name of the company that owns the package.
URL	The website address for the owner of the extension.
Email Address	The package owner's support or contact email address.

Extension Configuration

Each extension type has different configuration options that are managed in the Extensions Settings section at the top of each extension detail page. Extension types can have very detailed configurations that are necessary to use the extension, like the module extension type. Other extension types, like the library extension, have no configuration outside of the common package settings. This section explains the configuration options available to each extension type.

Skin and Container Extension Configuration

The skin and container extensions are managed using the same interface and available options. Once you create your base skin or container extension, the only other configuration options are to change the package name and make the extension aware of the skin and container files. [Figure 18.3](#) shows the `Skinning Demo` example skin. With skin and container extensions, making DNN aware of the skins or containers in the extension is unnecessary because skins are made available through the convention of having them in the correct folder under the host- or site-specific Skins or Containers folder. For more information on the folder structure of skins and containers, see [Chapter 17](#).



[Figure 18.3](#)

Module Extension Configuration

A DNN module has to be configured correctly before it can be added to a page and used in a site. The extension settings for a module are broken up into two sections. The first section is where host administrators can manage the folder name, business controller class, and whether the module should be available only to specified sites. [Table 18.2](#) explains the configuration options in the first section for module extensions.

Table 18.2 Configuration Options for Module Extensions

Attribute	Purpose
Module Name	Used within page and site templates to specify the module type. Must be unique.
Folder Name	The name of the folder within the <code>DesktopModules</code> folder where the module's controls are stored.
Module Category	A category that can be assigned to the module. The module's category is used by the control panel to group modules and make them easier to find. The Taxonomy page under the Host menu can be used to create new categories.
Business Controller Class	The fully qualified name of the class that implements any module features interfaces (<code>IPortable</code> , <code>ModuleSearchBase</code> or <code>ISearchable</code> , and <code>IUpgradeable</code>).
Is Portable?	Read-only setting that indicates whether the module exposes import/export functionality through implementing the <code>IPortable</code> interface.
Is Searchable?	Read-only setting that indicates whether the module implements the <code>ModuleSearchBase</code> base class (or older <code>ISearchable</code> interface) necessary to expose information to the DNN search index engine.
Is Upgradable?	Read-only setting that indicates whether the module supports the <code>IUpgradeable</code> interface allowing it to run custom code when installed and upgraded.
Module Sharing	Indicates whether the module can share instances of itself across multiple sites.
Is Premium Module?	Sets whether, on site creation, a module should automatically be available to be used by administrators of that site (premium modules are <i>not</i> automatically available).

Assigned Premium Modules	Sets which site have access to premium modules.
--------------------------	---

The second section for configuration is for managing Module Definitions. Most modules will have a single Module Definition for each module extension. A Module Definition is the parent entity for controls to be used to add functionality with a DNN module. If your module needs to have more than one control added to the page for the module to function correctly, you can achieve this through adding multiple Module Definitions. When a module with multiple Module Definitions is added to a page, a module instance is added for each definition.

By adding multiple Module Definitions, you can split functionality that is necessary into different control parts that can be laid out with container and security settings that are independent of each other. For the blog module example, this hides the complexity of having to add five modules to the page to have the module behave correctly and allows site administrators to place the archive, search, and blog list in whichever order meets the needs and design requirements of the organization.

After you create a Module Definition for your module, you need to add controls to it. Using controls, module developers add the default `view`, `settings`, and `edit` controls and then any other custom controls the module needs. Follow these steps to set up the `Wrox.Suggestions` module from the module development chapters earlier in the book:

1. Click the Create New Extension link on the Host extensions page.
2. Select Module from the drop-down list.
3. Enter **Wrox.Suggestion** for Name.
4. Enter **Suggestions Module** for Friendly Name.
5. Enter **The Suggestions demo module** for Description.
6. Set Version to 2.0.0.
7. Once your screen matches [Figure 18.4](#), click Next.
8. On the next screen, enter **Wrox.Suggestion** for Folder Name to let the extensions management know where to find the module controls.
9. Choose **Common** for Module Category. This sets the category the module appears in in the Add Module control bar when using the module after it is

completed.

10. Enter **WROX.Modules.Suggestion.SuggestionController**, **WROX.Modules.Suggestion** for the Business Controller Class.
11. Once your screen matches [Figure 18.5](#), click Next.
12. Leave the owner details blank and click Next again to complete the New Extension Wizard.

AWESOME

DNN 7 > Extensions > New Extension Wizard

Create New Extension

The Create New Extension Wizard allows you to create new Extensions. Enter the information requested in each step. Please select the type of Extension to create.

Select Extension Type:

In the first step please provide a unique name for the new Extension, - it is recommended that you use the format CompanyName.Name to avoid potential clashes if you intend to distribute the extension. In addition, you should provide a friendly name for the Extension, a description for the Extension and you should select a version (or the default version No. 0.0.0 will be used)

Name: *

Friendly Name: *

Description:

Version:

[Figure 18.4](#)

The screenshot shows the 'DNN 7 > Extensions > New Extension Wizard' window. The 'Module Specific Details' section is active, with the instruction: 'In this step please provide any module specific information.' The 'Module Settings' section contains the following fields:

- Module Name:
- Folder Name:
- Module Category:
- Business Controller Class:
- Dependencies:
- Permissions:
- Is Portable?: False
- Is Searchable?: False
- Is Upgradable?: False
- Module Sharing:
- Is Premium Module?:

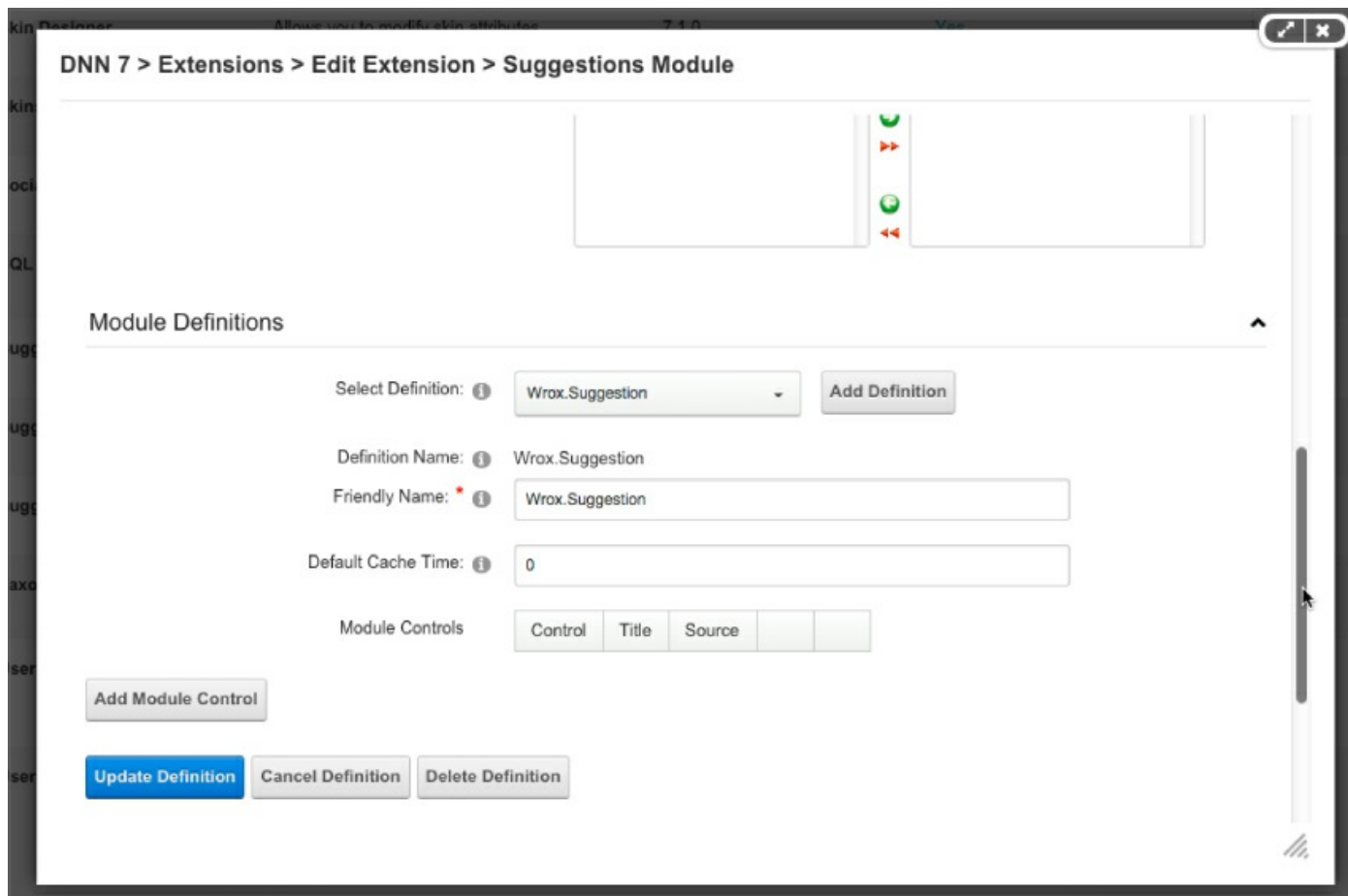
At the bottom, there are 'Next' and 'Return' buttons.

Figure 18.5

Now that your module extension is created, click the pencil icon to edit the details for that module. Follow these steps to create the Module Definitions for the Suggestions module:

1. Locate and expand the Module Definitions section.
2. Click Add Definition; then type **Wrox.Suggestion** in the Definition Name and Friendly Name textboxes.
3. For read-only modules, Default Cache Time can be changed from 0 to some number of seconds to have the output of the module cached by default. Default Cache Time can also be changed to -1 to indicate that the module does not support output caching at all.

- Click Create Definition to create the Module Definition. Once you have created the Module Definition, a new interface is available below the definition area for managing module controls, as shown in [Figure 18.6](#).



[Figure 18.6](#)

Click the Add Module Control button to add a new control to the Module Definition. The Edit Module Control screen allows developers to define the controls to operate the module. [Table 18.3](#) explains what each field does.

[Table 18.3](#) Module Control Definition Attributes

Name	Purpose
Key	Sets the URL parameter to load the control. Leave this blank for the default view control.
Title	For controls other than the default view control, this displays as the module title when the control loads.
Source Folder	The folder in the DNN installation containing the ASCX user control.

Source	The ASCX user control to use for displaying the interface to the user.
Type	Sets the security level type of the control.
View Order	Sets the order in which the control is displayed in the list of controls on the Module Definition page.
Icon	Sets the icon to be displayed with the module title when the control is loaded. Like the module title, this is used only when loaded as a control other than the default view control.
Help URL	The URL that should be used for the help link in the module actions menu. If this is left blank, it will use the context-sensitive Online Help at the DNN website.
Supports Popups	Indicates whether DNN can load the control in a pop-up, if the site is configured to use pop-ups.
Supports Partial Rendering	This tells DNN to wrap the user control in an UpdatePanel control.

To continue building the suggestions Module Definition, add the three controls according to [Table 18.4](#). For the Source Folder field in each module control, select the folder already created for the Suggestions module.

Table 18.4 Wrox.Suggestion Module Definitions Module Controls

Key	Control Source	Title
[Blank]	DesktopModules/WROX.Suggestion/Suggestion.ascx	[Blank]
Edit	DesktopModules/WROX.Suggestion/EditSuggestion.ascx	Edit Content
Settings	DesktopModules/WROX.Suggestion/Settings.ascx	Suggestion Settings

To finish setting up the Suggestions module, add a second Module Definition named **Wrox.Suggestion.Display** with a single view control pointing to `DesktopModules/WROX.Suggestion/DisplaySuggestions.ascx`. Click Update Extensions to save your changes to the extension.

Skin Object Configuration

The Skin Object extension type is used to extend the DNN skinning engine by

adding ASCX user controls that can be referenced by skins using tokens and the new Skin Object format. The Skin Object configuration attributes are explained in [Table 18.5](#).

Table 18.5 Configuration Options for Skin Object Extensions

Setting	Purpose
Control Key	The unique key used to refer to the Skin Object from HTML-based skins. This is the value used in the codebase parameter for Skin Objects.
Control Src	The location of the ASCX user control for the Skin Object.
Supports Partial Rendering	This tells DNN to wrap the user control in an UpdatePanel control to intercept postbacks.

Provider Extension Configuration

Providers are pluggable pieces of functionality used to perform base behaviors and framework building blocks for DNN. Providers have no custom configuration interface. All management for providers is handled through the package configuration file.

Authentication System Configuration

The Authentication System extension type is a special type of provider that controls authentication and authorization for resources belonging to a site. The options available for configuration to the Authentication System extension type are listed in [Table 18.6](#).

Table 18.6 Configuration Options for Authentication System Extensions

Setting	Purpose
Authentication Type	This is the unique key to set the name of the authentication system.
Login Control Source	This is the ASCX user control that should process a login.
Logoff Control Source	This is the ASCX user control that should process a logoff.
Settings	This is the ASCX user control that manages the configuration

Control Source Enabled	options for an authentication provider. Sets whether this authentication provider should be available.
------------------------	---

Core Language Pack Extensions Configuration

The Core Language Pack extension allows host administrators to install and manage sets of local resource files containing strings to translate interface text for different languages that are supported. The New Extension Wizard requires you to choose the language with which the language pack should be associated (in order to be selected, the language needs to already have been enabled via the Languages page in the Admin menu). The extension configuration can then be updated to change the language.

Extensions Language Pack Extensions Configuration

The Extensions Language Pack extension type creates a way to distribute language packs for modules, providers, and other extension types. The New Extension Wizard asks for the language and package with which to associate the language pack. The extension configuration allows changing the language and the package.

Using the Wizard to Create Packages

DNN 5's enhancements to the Package Creation Wizard dramatically simplified the process of packaging extensions for distribution, and this is still in use for DNN 7. The Package Creation Wizard takes an existing extension and creates the properly formatted package definition file and zips the selected files to create the distributable package. To create a package, click the Create Package link at the bottom of the extension details page. Clicking the Create Package link starts the Create Package Wizard. The following lists the steps in the Package Creation Wizard:

1. **Review Package Information.** The first step in the Create Package Wizard is a simple review of the extension from which you have chosen to create a package. The two options on this screen are Use Existing Manifest and Review Manifest. The Use Existing option bypasses the step to create a package manifest. The Review Manifest option tells the wizard to display an editable version of the package manifest before the package is created.
2. **Choose Files to Include.** Most packages include the second step to allow you to select files to include in the extension package. Select the folder to scan and click Refresh File List to populate a list of all files and subfolders' files. To exclude a file that was selected, simply delete the line from the textbox.
3. **Choose Assemblies to Include.** Packages that include server-side functionality, like modules, providers, and authentication systems, will show the third possible step in the wizard for including assemblies. To include an assembly, simply add a line for each DLL to include. The wizard assumes that the root folder for assemblies is the `bin` folder in the root of the website, so there is no need to add the folder name to the beginning of the filename. [Figure 18.7](#) shows the assemblies entry for the `Wrox.Suggestion` module example from this and earlier chapters.
4. **Create Manifest.** All extension types require a manifest file. The Create Manifest step in the wizard allows you to review the manifest file created by the wizard before it is included in the package in the last step of the wizard. The manifest is displayed in an editable textbox so that you can edit any elements in the manifest before creating your package.
5. **Create Package.** The final step tells the wizard what to name the manifest file and where to save the package. Manifest files should have the

extension `.dnn` and the archive filename should use the `.zip` extension because it is a compressed zip folder. Regardless of whether you choose to create the manifest or create the archive, the package will be added to the database and the manifest will be saved to the package archive. All packages that are created will be placed in a subfolder of the `\Install` folder in the DNN instance, based on the type of extension (i.e., module packages will be placed in the `\Install\Module` folder).

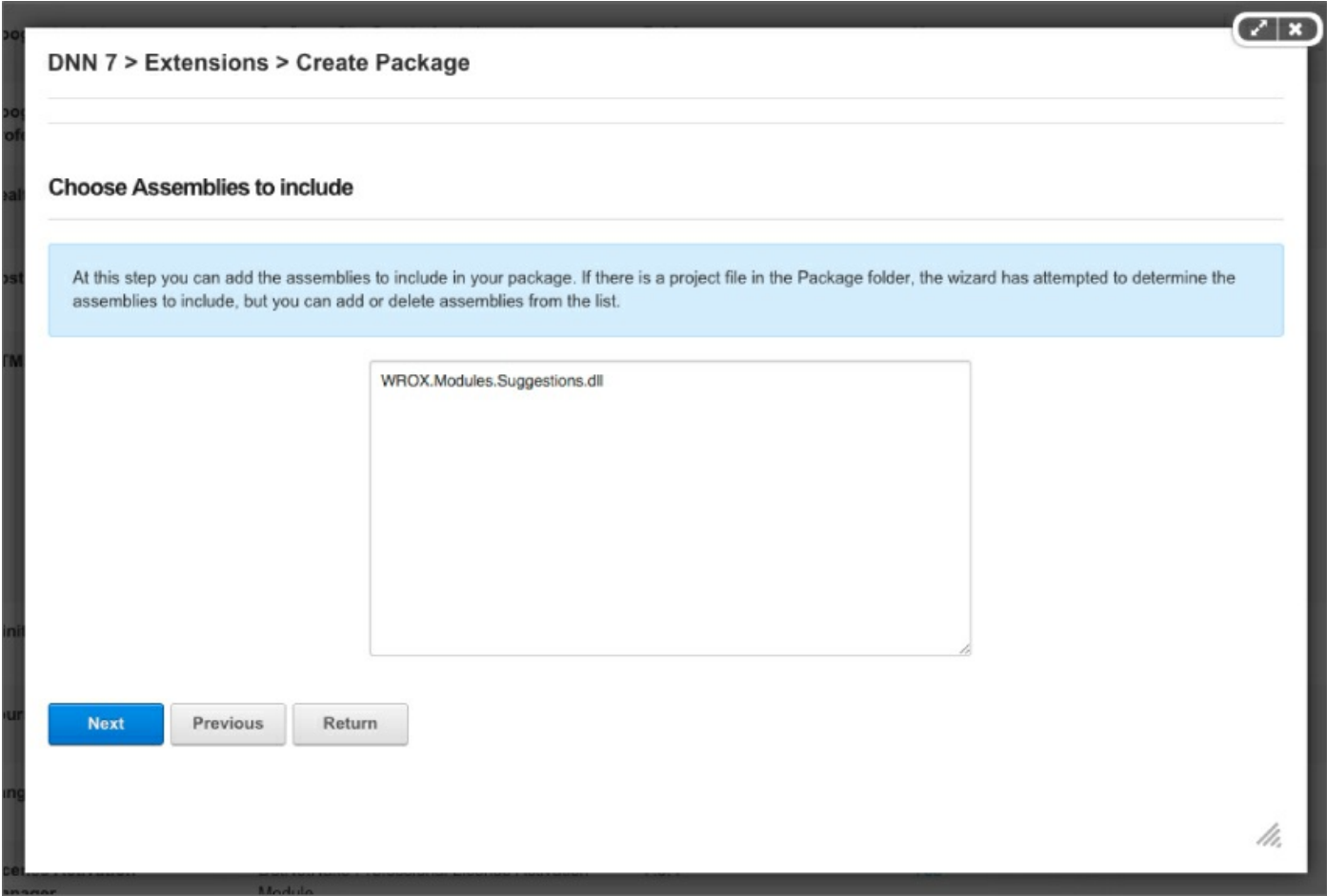


Figure 18.7

Building Packages with Manifest Files

In the previous section, you learned that the Package Creation Wizard is a simple way to package skins, modules, and other extensions for distribution. Though the wizard simplifies the process of packaging and creating the manifest for packages to be distributed, there will always be times that you'll need to modify the package and the manifest manually to get the exact behavior you require for your installation process. This section explains version 5 of the DNN manifest file format and provides examples for common tasks using the manifest file.

Manifest Packages

Packaging an extension without using the Package Creation Wizard is done by adding the files from your extension into a compressed zip folder and telling the installer how to use the files with an XML-based manifest file. An installable extension is referred to in the manifest as a *package*. The architecture of the DNN installer provides for installing multiple packages into a single installation by adding multiple packages under the packages node. This could allow a developer to distribute an install package with libraries, skins, modules, and a provider in a single install. The package information begins at the `dotnetnuke/packages/package` element within the XML structure. [Table 18.7](#) explains the general package elements and attributes.

Table 18.7 Package Elements and Attributes

Name	Path	Purpose
Package Name	<code>dotnetnuke/packages/package/@name</code>	This name must be unique because it is used in site and page templates to refer to modules and extensions.
Type	<code>dotnetnuke/packages/package/@type</code>	Setting that

		<p>instructs DNN whether to treat the extension as a module, skin, provider, or any other custom extension type.</p>
Version	<code>dotnetnuke/packages/package/@version</code>	<p>Version number for the package. Versions should include three numeric parts, separated by dots (e.g., 1.23.0).</p>
Friendly Name	<code>dotnetnuke/packages/package/friendlyName</code>	<p>This setting is for the name to display to end users. It is used in drop-downs, lists, and the control panel for modules.</p>
Description	<code>dotnetnuke/packages/package/description</code>	<p>Setting to display a short description.</p>

		of the extension in the list of extensions on the Host and Admin extensions pages.
Icon File	<code>dotnetnuke/packages/package/iconFile</code>	Path to the icon (i.e., a 32x32 image file) for the extension. It is used in lists and the control panel for modules.
Owner Name	<code>dotnetnuke/packages/package/owner/name</code>	The owner of the package.
Organization	<code>dotnetnuke/packages/package/owner/organization</code>	The name of the company that owns the package.
URL	<code>dotnetnuke/packages/package/owner/url</code>	The website address for the owner of the extension. This field can contain HTML, if you would like the URL to be a link.
Email Address	<code>dotnetnuke/packages/package/owner/email</code>	The package owner's support or

		contact email address. This field can contain HTML, if you would like the email address to be a link.
License	<code>dotnetnuke/packages/package/license</code>	HTML representing the license (EULA) of the extension. Alternatively, the license can be provided via an external file through the <code>src</code> attribute described next.
License Source	<code>dotnetnuke/packages/package/license/@src</code>	When specified, overrides the license HTML with the content of a file in the package
Release Notes	<code>dotnetnuke/packages/package/releaseNotes</code>	HTML representing notes about the current

		version of the extension. Alternatively the notes can be provided via an external file through the <code>src</code> attribute described next
Release Notes Source	<code>dotnetnuke/packages/package/releaseNotes/@src</code>	When specified, overrides the release note HTML with the content of a file in the package

Continuing with the example from earlier in this chapter, the following code shows the package description part of the manifest file for the `Wrox.Suggestion` module:

```
<dotnetnuke type="Package" version="5.0">
  <packages>
    <package name="Wrox.Suggestion" type="Module" version="2.0.0">
      <friendlyName>Suggestions Demo Module</friendlyName>
      <description>The suggestions demo module.</description>
      <owner>
        <name>Wrox</name>
        <organization>Wiley and Sons Publishing</organization>
        <url>http://p2p.wrox.com</url>
        <email>support@wrox.com</email>
      </owner>
      <license>Suggestions is provided as a demonstration module for
        DNN as part of the Professional DNN 7 book from Wrox.
        It may not be redistributed or sold. </license>
      <releaseNotes>Version 2.0 is dependant upon DNN 7. Any
questions
        can be submitted through p2p.wrox.com.
</releaseNotes>
```

```
<components>
    ...
</components>
</package>
</packages>
</dotnetnuke>
```

Package Components

The building blocks of an extension manifest file are the components that are included in the package. A component is a logical grouping of a part of the package to be installed. For most package types, there is a component type that matches the package type. For example, a skin package type will have a skin component type, and a module package type will include a module component type. The real advancement in the package manifest file is that common installation types like SQL scripts, assemblies, and generic files have been broken out so that they can be used across all package types.

File Components

The File component is a common component type that adds files to the DNN installation. This component type can be used in all package types. The addition of the File component type is an advancement over previous versions of DNN because it allows developers to install files into other parts of the website other than the `DesktopModules` folder. Using the File component type, set the `basePath` attribute to set where the group of files should be installed and include a list of files to install with the package. The following example shows a File component:

```
<component type="File">
  <files>
    <basePath>DesktopModules\Wrox.Suggestion</basePath>
    <file>
      <path>app_localresources</path>
      <name>suggestion.ascx.resx</name>
    </file>
    <file>
      <name>displaysuggestions.ascx</name>
    </file>
  </files>
</component>
```

In the example, you can see that the `basePath` node sets the folder into which all listed files are relative. Within the `files` node, each file has a name and optional path that is relative to the `basePath` root folder. Additionally, a

`sourceFileName` element can also be specified alongside the `name` element, which specifies the name of the file in the package, if it's different than the name the file should have when installed.

Many of the other components are based on the file component, including Assembly, Cleanup, JavaScriptFile, Language, ResourceFile, Script, Skin, Container, and Widget. As a result, they share these same basic elements.

Assembly Components

The Assembly component is used for installing DLLs into the `bin` folder in your DNN installation. The Assembly component is very similar in structure to the File component. Set the `basePath` to `bin` and list the assemblies using the `assembly` node type. One additional element used by the Assembly component is the `version`, which is used to avoid clashes between different extensions using the same assembly. When installing an Assembly component, if a higher version of the assembly has already been installed, it is skipped. The `version` element is optional and defaults to the version of the package if it isn't specified.

```
<component type= "Assembly">
  <assemblies>
    <basePath>bin</basePath>
    <assembly>
      <name>WROX.Modules.Suggestion.dll</name>
    </assembly>
    <assembly>
      <name>WROX.Modules.Suggestion.SqlDataProvider.dll</name>
    </assembly>
  </assemblies>
</component>
```

Script Components

The Script component is used for executing install and uninstall SQL scripts to configure the database storage and methods needed to support the extension in the package. SQL script filenames should use the name of the data provider as their file extension, which will be `.SQLDataProvider` unless a custom data provider is being used. Simply using `.sql` as the file extension is also supported but does not indicate the data provider for which the script is written.

In previous versions of the installer, versioning was managed using the convention of the filename. In this version of the installer, the `version` node

in the script element should correspond to the version number of the package being installed. The `version` node tells DNN when to run the script. By versioning your `SqlDataProvider` files consistently with the version number of the package, the DNN installer will run the SQL scripts as database upgrade scripts depending upon the currently installed version. When installing a newer version of an existing package, the installer will only run scripts with version numbers after the current version. For example, if you have version `01.00.00` of a module installed and you install an upgrade for version `01.00.01`, if the package includes scripts with the version set to `01.00.00` and `01.00.01`, the DNN installer would run only the second SQL script. If the package has never been installed, then all scripts will be executed in order of version number. This functionality protects existing tables from accidentally being deleted or cleared with the initial creation scripts and ensures that table update scripts run only once. The following example shows the Script component type from the `Wrox.Suggestion` module:

```
<component type="Script">
  <scripts>
    <basePath>DesktopModules\Wrox.Suggestion</basePath>
    <script type="Install">
      <name>02.00.00.sqldataprovider</name>
      <version>02.00.00</version>
    </script>
    <script type="UnInstall">
      <name>uninstall.sqldataprovider</name>
      <version>2.0.0</version>
    </script>
  </scripts>
</component>
```

Two special types of script files are supported, for the cases where a unique script is required to be run on first install or a script is required when upgrading to a later version. These are used by naming convention and included into the `scripts` node as normal.

For a script that is to be run on first install, before any other scripts, use the `name` `install.SqlDataProvider`.

For a script that is to be run on upgrade of an existing extension, use the `name` `upgrade.SqlDataProvider`. This script will be run as the final script when the upgrade is run, regardless of order in the `scripts` node.

Cleanup Components

The Cleanup component provides a mechanism for deleting files and assemblies that are no longer needed for a version of a package. In previous versions of the installer, developers could instruct DNN to remove a list of files by providing a file with the naming convention `[version number].txt` that included a list of files to delete when installing that version. Like the Scripts component discussed earlier, the installer runs the cleanup for all versions between the current version of the module and the version being installed.

Because the installer introduced in DNN 5 is based upon configuration over convention, developers are required to declare explicitly the files to be removed in the manifest file. Files can be listed within the Cleanup component or referred to through a `src` attribute. The version must be set within the `version` attribute of the Cleanup component for the installer to know which cleanup tasks should be performed to upgrade an installed version to a current version of the package. The following examples show how to mark files for deletion:

```
<component type="Cleanup" version="02.00.00">
  <files>
    <file>
      <path>DesktopModules\WROX.Suggestion\images</path>
      <name>oldImage.jpg</name>
    </file>
  </files>
</component>
```

Alternatively, you can point your Cleanup element to a file included in your package like in the following example:

```
<component type="Cleanup" version="02.00.00" src="02.00.00.txt"/>
```

The contents of the referenced `02.00.00.txt` file would simply be a list of file paths, such as the following example:

```
DesktopModules\WROX.Suggestion\images\oldImage.jpg
```

Config Components

The Config component gives developers the ability to manage changes to the `web.config` files from the install process. This feature can save users without experience from having to edit the `web.config` file when it is necessary for a package to be installed. It also protects commercial developers from the support calls and emails that will inevitably come from those inexperienced

users breaking their web.config when they incorrectly make the changes necessary for the package.

The Config component has an `install` section and an `uninstall` section to provide instructions on how to make the changes to web.config when the package is installed and how to remove or revert the configuration to its state before the configuration was changed (both the `install` and `uninstall` sections are required, even if they only have an empty `nodes` element). The following code shows how the Config component could be used to add an `appSetting` key to the `configuration` section of web.config and remove the same key when the package is uninstalled:

```
<component type="Config">
  <config>
    <configFile>web.config</configFile>
    <install>
      <configuration>
        <nodes>
          <node path="/configuration/appSettings" action="add"
key="key"
collision="overwrite">
            <add key="WROX.Suggest.Type" value="Module"/>
          </node>
        </nodes>
      </configuration>
    </install>
    <uninstall>
      <configuration>
        <nodes>
          <node
path="/configuration/appSettings/add[@key='WROX.Suggest.Type']"
action="remove"/>
        </nodes>
      </configuration>
    </uninstall>
  </config>
</component>
```

Each node in the nodes element defines an action to be performed on the config file. Using the path, the section of the .config file to be modified is selected using an Xpath expression. [Table 18.8](#) lists the actions for the `node` element and how it is used to manage web.config.

Table 18.8 Node Action Types

Action Type	Purpose

<code>add</code>	Adds the content within the node element to the location selected in the path
<code>insertbefore</code>	Inserts the supplied content before the selected node in the path
<code>insertafter</code>	Inserts the supplied content after the selected node
<code>remove</code>	Removes the node selected in the path
<code>removeattribute</code>	Removes the attribute selected in the path
<code>update</code>	Updates the node selected in the path with the supplied content
<code>updateattribute</code>	Updates the selected attribute

Besides `path` and `action`, there are a few other node attributes that control how the node action is performed on the selected node. The `key` attribute instructs DNN which attribute in the XML node to use to compare nodes to look for duplicates. In the earlier example, the `key` was set to `key` to let the installer know that `appSettings` that have the same `key` attribute are duplicates. In the provider config type, you would change the `key` to `name`. If there is no attribute indicating a duplicate, but the element itself would be a duplicate, the `targetpath` attribute can be used to supply an Xpath expression pointing to the element.

The last attribute for the `node` element is `collision`. `collision` tells the installer how to handle duplicates that it finds using the `key` attribute. You can supply `save` to have the installer save a commented-out version of the old node, `overwrite` to replace the old node with the one you supply, or `ignore` to leave an existing node as is.

Since the Config component does not specify a version, it runs every time the extension package is installed. Therefore, it's important to make sure that your configuration actions are safe to occur multiple times. Most of the time, the `update` or `updateattribute` action is the correct choice to use, making sure that a `key` or `targetpath` is specified to avoid duplicates.

Module Components

The Module component is used to install modules as part of a package. The Module component is built with a serialized XML `DesktopModule` object type. The layout of the Module component type should look very familiar if you have worked through the Create Package Wizard for building a module

package. Using the `desktopModule` node, the storage location, Module Definitions, and controls are defined in XML.

```
<component type="Module">
  <desktopModule>
    <moduleName>Wrox.Suggestion</moduleName>
    <foldername>Wrox.Suggestion</foldername>

    <businessControllerClass>WROX.Modules.Suggestion.SuggestionController,
    WROX.Modules.Suggestion</businessControllerClass>
    <supportedFeatures>
      <supportedFeature type="Portable"/>
      <supportedFeature type="Searchable"/>
    </supportedFeatures>
    <moduleDefinitions>
      <moduleDefinition>
        <definitionName>Wrox.Suggestion</definitionName>
        <friendlyName>Wrox.Suggestion</friendlyName>
        <defaultCacheTime>0</defaultCacheTime>
        <moduleControls>
          <moduleControl>
            <controlKey/>
          </moduleControl>
        </moduleControls>
      </moduleDefinition>
    </moduleDefinitions>
    <controlSrc>DesktopModules/WROX.Suggestion/Suggestion.ascx</controlSrc>

    <supportsPartialRendering>False</supportsPartialRendering>
    <controlTitle/>
    <controlType>View</controlType>
    <iconFile/>
    <helpUrl/>
  </desktopModule>
</component>
```

Developers who use the website project model for developing modules and have business logic included in the `App_Code` folder can include the `.cs` or `.vb` files in their installation packages with the Files component explained earlier in the chapter.

Upgradeable Interface and the `eventMessage` Section

The version control and management within the Scripts and Cleanup components of the extension manifest provide a rich API for developers to

provide upgrade paths to consumers of their modules, but there are times when these tools are no substitute for access to the full .NET Framework. The DNN extensions manifest accounts for the need to execute ASP.NET code to execute upgrade operations that are not covered with file copy, file delete, or T-SQL scripts with the `eventMessage` section within the Module component. When the module install process runs, the last action that the installer performs is installing DLL assemblies into the `bin` folder of the website. Because adding assemblies to the `bin` of an ASP.NET website always causes the application to restart, access to the current thread is lost with that application restart and no more server-side code can be run until ASP.NET recompiles and starts the worker process. The problem that this presents for server-side upgrade code is that the upgrade code is included in the new assembly, which cannot be run until the application restarts, at which point the upgrade process is no longer running. To address this problem, DNN uses the `eventMessage` process of storing an upgrade action to be run on `Application_Start` event. The following example shows a sample `eventMessage` section:

```
<component type="Module">
  <desktopModule>
    <moduleName>WROX.Modules.UpgradeApp</moduleName>
    <foldername>WROX.Modules.UpgradeApp</foldername>

    <businessControllerClass>WROX.Modules.UpgradeApp.WroxBusinessControlle
WROX.Modules.UpgradeApp</businessControllerClass>
    <supportedFeatures/>
    <moduleDefinitions/>
  </desktopModule>
  <eventMessage>
    <processorType>DotNetNuke.Entities.Modules.EventMessageProcessor,
otNetNuke</processorType>
    <processorCommand>UpgradeModule</processorCommand>
    <attributes>

    <businessControllerClass>WROX.Modules.UpgradeApp.WroxBusinessControlle
WROX.Modules.UpgradeApp</businessControllerClass>
    <desktopModuleID>[DESKTOPMODULEID]</desktopModuleID>

    <upgradeVersionsList>01.00.00,01.00.01,02.00.01</upgradeVersionsList>
    </attributes>
  </eventMessage>
</component>
```

When the module is upgraded, DNN uses the `eventMessage` section of the Module component to store an event in the `EventMessage` table to run the next time the application starts. When the `EventMessage` is run, the business controller's `UpgradeModule` event runs once for every version listed in the `upgradeVersionsList` node. The method is executed in the order in which it is listed, regardless of the version that is currently installed or the version to which it is being upgraded. That means that the upgrade process runs differently than the Scripts upgrade process in that it does not keep track of the version it is upgrading to or from. Within the `UpgradeModule` event, the logic will have to account for the version number being passed to it and execute the appropriate upgrade operations based upon that. Following is an example `UpgradeModule` using a `switch` statement to execute code specific to the version that is being upgraded:

```
namespace WROX.Modules.UpgradeApp {
    public class WroxBusinessController:
DotNetNuke.Entities.Modules.IUpgradeable {
        #region IUpgradeable Members
        public string UpgradeModule(string Version) {
            switch (Version) {
                case "01.00.00":
                    //Upgrade from 01.00.00
                    return (string.Format("Successfully upgraded from version
{0}"
, Version));
                case "01.00.01":
                    //Upgrade from 01.00.01
                    return (string.Format("Successfully upgraded from version
{0}"
, Version));
                case "02.00.01":
                    //Upgrade from 02.00.01
                    return (string.Format("Successfully upgraded from version
{0}"
, Version));
                default:
                    return (string.Format("Version {0} upgrade not supported.",
Version));
            }
        }
    }
    #endregion
}
```

The message returned from the `UpgradeModule` method is logged for each upgrade event and is viewable on the Event Viewer page in the Admin menu.

Skin and Container Components

Skins and containers are created easily and efficiently with the Create Package Wizard. Designers and developers should not generally need to create a manifest file from scratch for skins and containers. All files in the skin or container package are listed under the respective `skinFiles` or `containerFiles` node. The following code shows example skin and container component types:

```
<component type="Skin">
  <skinFiles>
    <skinName>SkinningDemo</skinName>
    <basePath>Portals\_default\Skins\SkinningDemo</basePath>
    <skinFile>
      <name>htmlmethod.ascx</name>
    </skinFile>
    <skinFile>
      <name>htmlmethod.jpg</name>
    </skinFile>
    <skinFile>
      <name>htmlmethod.css</name>
    </skinFile>
    <skinFile>
      <name>htmlmethod.doctype.xml</name>
    </skinFile>
  </skinFiles>
</component>
<component type="Container">
  <containerFiles>
    <containerName>MinimalExtropy</containerName>
    <basePath>Portals\_default\Containers\MinimalExtropy</basePath>
    <containerFile>
      <path>images</path>
      <name>dnn-plus.png</name>
    </containerFile>
    <containerFile>
      <name>container.css</name>
    </containerFile>
    <containerFile>
      <name>title_blue.ascx</name>
    </containerFile>
    <containerFile>
      <name>title_blue.css</name>
    </containerFile>
    <containerFile>
      <name>title_blue.jpg</name>
    </containerFile>
  </containerFiles>
</component>
```

SkinObject Components

The SkinObject component tells DNN how to use the files included in the package to create the token for use in DNN skins. The SkinObject component does not include the actual ASCX files that are used for the SkinObject; it only maps the token to the user control used to render the SkinObject. You should also include a File component type to include the ASCX files and any other supporting images or files needed to render the SkinObject.

```
<component type="SkinObject">
  <moduleControl>
    <controlKey>ACTIONBUTTON</controlKey>
    <controlSrc>Admin/Containers/ActionButton.ascx</controlSrc>
    <supportsPartialRendering>False</supportsPartialRendering>
  </moduleControl>
</component>
```

Language Pack Components

In order to instruct the DNN installer how to use your language pack, you will provide a group of `languageFiles` elements for each language that is in the install package. Unlike the Create Wizard shown earlier in the chapter, the language does not need to already be enabled before installing the component. The information in the manifest will be used to set up the language, if it is not already set up. To do that, the manifest will specify the language code, display name, and fallback language. The remainder of the component type is filled with `languageFile` elements. `languageFile` elements should all have a `path` element and a `name` element for each RESX file, as shown in the following code example. The Extension Language package should be created with the same structure as the `CoreLanguage` element except with `ExtensionLanguage` as the component type key and a `package` element to declare which extension it is translating.

```
<component type="CoreLanguage">
  <languageFiles>
    <code>en-US</code>
    <displayName>English (United States)</displayName>
    <fallback/>
    <languageFile>
      <path>admin\controlpanel\app_localresources</path>
      <name>classic.ascx.resx</name>
    </languageFile>
  </languageFiles>
</component>
```

Authentication System Components

The Authentication System component type is a serialization of the inputs from the Create New Extension Wizard to control the login, logoff, and settings control sources. The Authentication System settings and Assembly component types can be mostly generated using the wizard, but web.config changes will have to be created manually using the Config component type explained earlier in this chapter. The following code shows an example Authentication System component:

```
<component type="AuthenticationSystem">
  <authenticationService>
    <type>LiveID</type>

    <settingsControlSrc>DesktopModules/AuthenticationServices/LiveID/Setti

      </settingsControlSrc>

    <loginControlSrc>DesktopModules/AuthenticationServices/LiveID/Login.as

      </loginControlSrc>

    <logoffControlSrc>DesktopModules/AuthenticationServices/LiveID/Logoff.

      </logoffControlSrc>
    </authenticationService>
  </component>
```

Resource File Components

While it is possible to copy all of the individual files for an extension to the correct location by using File component entries, this approach can become verbose and error prone when complexity increases. It can be difficult to track new files added to an extension between packaging, and it is time-consuming to debug missing files.

The solution to this problem is the Resource File component. This component acts as a File component for a compressed folder of files destined for an extension folder. Typically for module extensions, all of the files to be copied into the `/DesktopModules/ExtensionName` path are compressed into a single .zip file. The Resource File component then locates this zip file within the overall Installation package and extracts all the extension files into the target folder. Only a single component entry is required in the manifest file, and any new files required for the extension are added to the zip file as required. No further changes are required for the manifest when new files are

added.

The following code shows a Resource File component in the manifest file. The contents of the `Wrox.Suggestion.zip` file are all of the `.ascx`, `.css`, `.js`, and other files destined for the `Wrox.Suggestion` folder. The zip file can contain subfolders, but all of the root level files should be in the root of the zip file.

```
<component type="ResourceFile">
  <resourceFiles>
    <basePath>DesktopModules\Wrox.Suggestion</basePath>
    <resourceFile>
      <name>Wrox.Suggestion.zip</name>
    </resourceFile>
  </resourceFiles>
</component>
```

Provider Packages

Provider packages do not have their own component types. Instead, components are made up of the Assembly, Config, and optionally the Script and File component types.

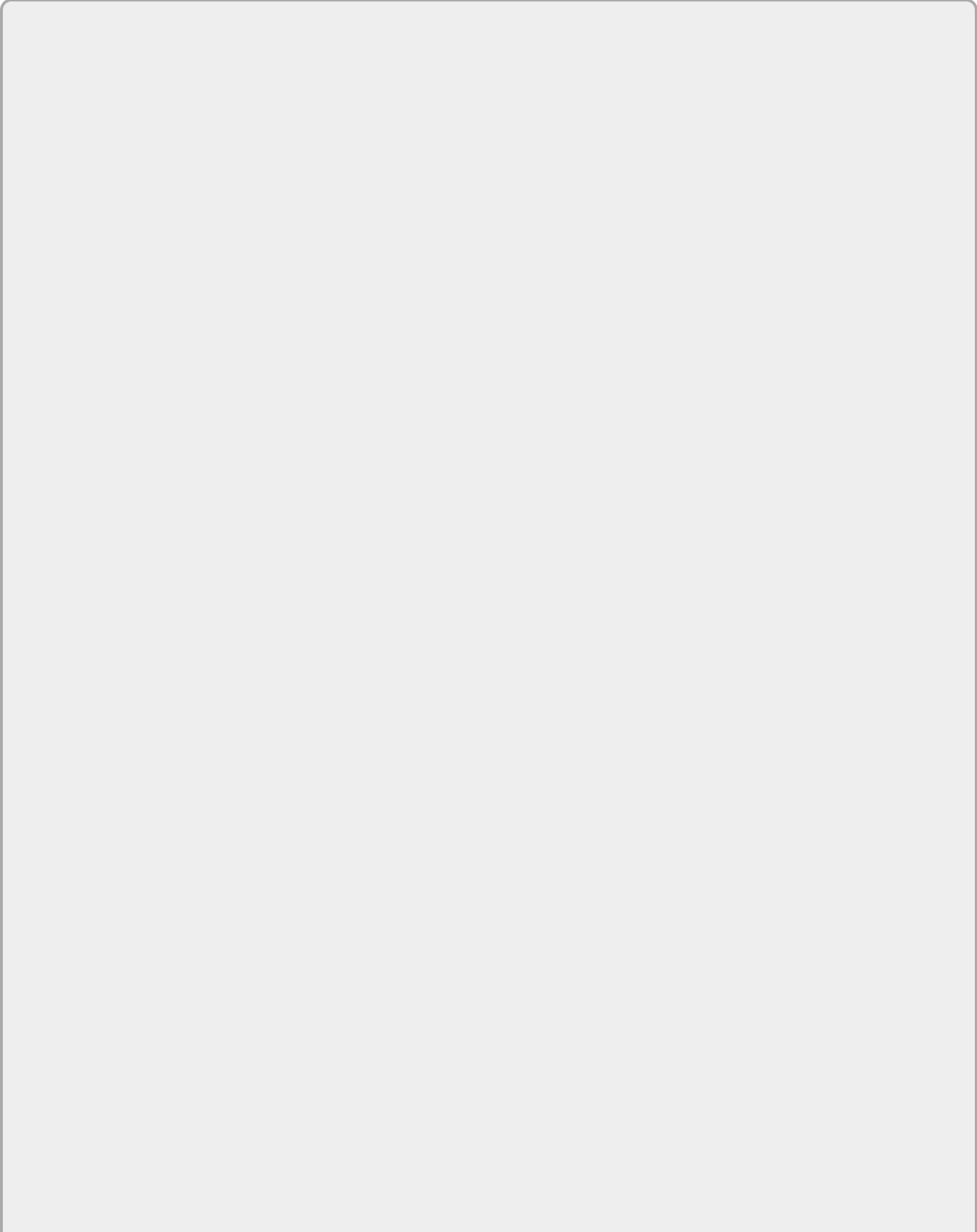
Library Packages

The Library Package extension is created to manage generic package types. Like the Provider package type, the Library Package does not have a component type. Using the Assembly, File, Script, and Config components, developers can install any kind of files or groups of files into a DNN installation. The Library Package type can be used to install and manage custom widgets, site templates, common DLLs, icon libraries, and anything else developers can think of to install.

Summary

In this chapter, you learned how to create and package extensions to the DNN Platform. You learned how to use the wizard to create install packages for the extensions that have an automated Package Creation Wizard. Lastly, you learned how to use the DNN manifest file format to install and manage extensions and providers for your installation.

Chapter 19
Commercial Philosophy



What You Will Learn in This Chapter

- Understanding the open core business model
- Recognizing DNN's technology advantage
- Identifying unique market conditions
- Leveraging an open source distribution model
- Building a strong brand identity
- Achieving commercial success

Similar to many open source projects, DNN was not created with commercial goals in mind. Rather, the commercial opportunity occurred as a result of the viral distribution and adoption of the open source project by individuals and organizations worldwide. Akin to planting a seed in fertile soil and giving it sufficient water and sunlight, the open source model allowed the DNN community to grow quickly and attract the right combination of users and vendors to create a powerful commercial ecosystem.

The Fundamentals

In order for a company to become commercially successful, it needs to have a unique advantage against its competition. Essentially this unique advantage has to be something that cannot be easily replicated. When it comes to companies utilizing an open source model, the unique advantage is nearly always something that appears to go somewhat against the spirit of the open source philosophy. This generally creates some tension in the community. A bit of tension can be healthy, but too much tension can have damaging effects on an ecosystem.

DNN Corp.'s commercial business is based on a concept known as the “open core” model. Essentially it involves the distribution of a fully functional open source product at the core and a separate commercial edition with proprietary features and professional support. The model attempts to create a unique advantage in the market for the vendor. However, open source equilibrium tends to govern the extent that the vendor can leverage this advantage, as the market tends to be self-governing and self-adjusting, thus avoiding the dilemma commonly referred to in economic theory as the “tragedy of the commons.”

If a vendor goes too far into the closed mode, it loses traction in the community and exposes itself to the threat of a fork. A *fork* is essentially another version of the open source core product that is developed and distributed by another organization. Nothing prevents others from developing the same proprietary features that the vendor has reserved for paying customers only and providing them as free, open source extensions or as an integrated part of a forked version. On the other hand, if the vendor goes too far into the open mode, it may weaken its business model and fail to generate enough revenue to sustain or grow its operations or provide sufficient stewardship or infrastructure to the open source community.

Ultimately, customers are the arbiters of success. If they are willing to pay for access to a product or service, a company has a decent opportunity to become successful and self-sustaining. However, an open core business model is viable only if the open source community is large, well established, and actively growing. This is because the conversion rates from the open source product to the commercial edition tend to be very modest. A conversion rate of 1% is not abnormal for many open core business models. A company can try to increase this conversion rate by employing a variety of business levers;

however, these levers generally tend to create tension in the open source ecosystem and increase the risk of a community revolt or fork.

It is important to note that the “open core” model is not the only approach for commercializing an open source project. In fact, there is a lot of controversy within the broader open source community related to the open core approach. The basic debate is whether differentiation at the product level can still maintain the symbiosis that is required to support the needs of all stakeholders in the ecosystem in the long term or if it ultimately leads to fragmentation. As a result, some open source vendors have chosen to avoid this dilemma and have adopted a model where there is only a single version of the open source product that is freely available to everyone, and they instead leverage ancillary options such as support, training, consulting, or hosting for commercialization. This is no panacea either, however, as it means that the vendor must compete with members of its own ecosystem for business opportunities.

So if the commercial viability of DNN Corp.'s open core business model was dependent on the existence of a large and healthy open source community, what were the factors that contributed to the creation of this ecosystem in the first place? Some might say that it was simply luck, that is, “being in the right place at the right time.” However, the reality is that it was a perfect storm of technology, market conditions, and a superior distribution model that enabled DNN to get traction and mature into a viable commercial business opportunity.

Technology

When you look at DNN from a technology perspective, you can see that there are a number of key attributes that allowed it to achieve rapid adoption. In general, it delivered a high amount of value at a very low cost, a combination that made the technology accessible to the masses and highly appreciated by its target audience.

The primary audience initially consisted of software developers familiar with Microsoft technology who were looking for an advanced web application framework. The frameworks that Microsoft provided tended to be stateless, consisting of a variety of disparate tools and APIs that software developers needed to figure out how to assemble themselves in order to create a functional web application. DNN was a stateful framework built on top of the Microsoft platform; with all of the parts already assembled in a modern, best practices architecture, providing a fully functioning web application. The strategy of delivering a pre-assembled, more functional and usable product to attract a large group of users has been played out numerous times over the history of the technology industry, dating all the way back to the dawn of the personal computer age and the success of the Apple I over the MITS Altair. Popular wisdom is that a product must deliver a 10X improvement over the competition in order for it to gain viral adoption. DNN benefitted from numerous case studies by organizations that claimed that they had achieved more than 10X productivity gains in their website development efforts by using the product.

Equally as important as identifying what DNN offered as a web application framework is identifying what it did not offer—this is what also differentiated it from other competitors. DNN was not a simple site builder. A variety of site builders already existed that focused on providing a simple user experience to non-technical users for developing web pages and publishing them to the web. The problem with the majority of these site builders is that they offered very limited opportunity for customization or personalization. DNN was also not an enterprise content management system (ECM). There were already many powerful ECM systems available that provided workflow, versioning, extensibility, and so on, in a closed system model and at a cost that made them affordable only to very large organizations. Both of these markets were red oceans with many companies focused on delivering similar solutions, which limited the opportunity for new competitors to enter. DNN filled a void

between these two types of competitors, a blue ocean that challenged the conventional wisdom at the time, by delivering a minimal set of core website management features and empowering software developers to easily extend the product with additional functionality. This was a totally new model for website development, not an incremental improvement to an existing model. Competitors specializing in non-Microsoft technology also recognized this opportunity, and soon an entirely new web content management market category began to emerge.

Market Conditions

At the time that DNN was first released, there was also a disruption in the Microsoft ecosystem, as the .NET Framework had just been introduced and many organizations were still grappling with their migration strategy. Because DNN was developed on the .NET Framework, it was viewed as modern and cutting edge, which attracted many early adopters who were looking to make the transition to the new framework. It was also a challenging time for developers who were experienced with Visual Basic, as all indications from Microsoft seemed to point to C# being the preferred development language of the future. With DNN based on Visual Basic.NET and fully open source, it provided Visual Basic developers with a familiar environment and Microsoft platform software developers in general with functional code examples to accomplish almost any development task they encountered in their business software projects. These factors allowed DNN to quickly dominate a large share of a small market, which later grew into a large opportunity as more organizations migrated to the latest versions of the Microsoft platform.

Distribution Model

The benefit of DNN's open source license cannot be overstated. Without it, there is no way that the product could have achieved such widespread distribution and adoption. It is extremely challenging for any software product to rely on traditional marketing and distribution methods and gain traction in the market. This is because the largest software companies tend to dominate the mainstream marketing and distribution channels, leaving limited opportunity for smaller companies to get noticed. In utilizing the open source model, a company concedes some value in their intellectual property but makes up for it by allowing any organization to use their software without restriction. If the open source product solves a large enough business problem, it has the ability to break free from the industry shackles and achieve widespread adoption and recognition. In addition, the open source distribution model is beneficial from a Customer Acquisition Cost (CAC) perspective, as it keeps the cost and resources associated to acquiring new customers to a minimum. This is precisely how DNN was able to experience viral growth and commercial success.

A network effect refers to the concept that a product or service becomes more valuable as more people use it. In the case of DNN, the freedom of the open source license allowed anyone to use the software without restriction. As the number of users increased, so did the activity in the online community support channels. DNN experts and evangelists shared their knowledge and opinions with other members of the community, creating a collaborative and empowering environment. As the size of the community grew, it also attracted companies that offered complementary services such as web hosting, custom development, and web design. These companies were essential in enabling new users to be successful with the open source product. Some of these companies took advantage of the opportunity to create extensions and offered them to customers as fully supported commercial products. So as the ecosystem grew and matured, the open source product became increasingly more valuable.

Branding

The widespread adoption of the open source project also cultivated a strong brand identity. Not only did the project name and logo receive universal recognition, but more importantly the open source project values and culture also became widely understood and respected in the industry. The concept of social entrepreneurship and “doing well by doing good” resonated with many people around the world and elicited an emotional response, resulting in a following that was almost cult-like in its loyalty and devotion.

This was not achieved purely from the distribution of the open source project but also from extensive marketing and communication efforts. The brand was emphasized on the website, in project documentation and collateral, and even within the software source code. The project values were regularly evangelized in newsletters, presentations, interviews, magazine articles, books, and conferences worldwide. Transparent communication channels allowed the community to feel like it had a personal relationship with the project and a direct channel for voicing feedback and influencing its direction. Over time the brand became one of the most valuable assets of the DNN open source project.

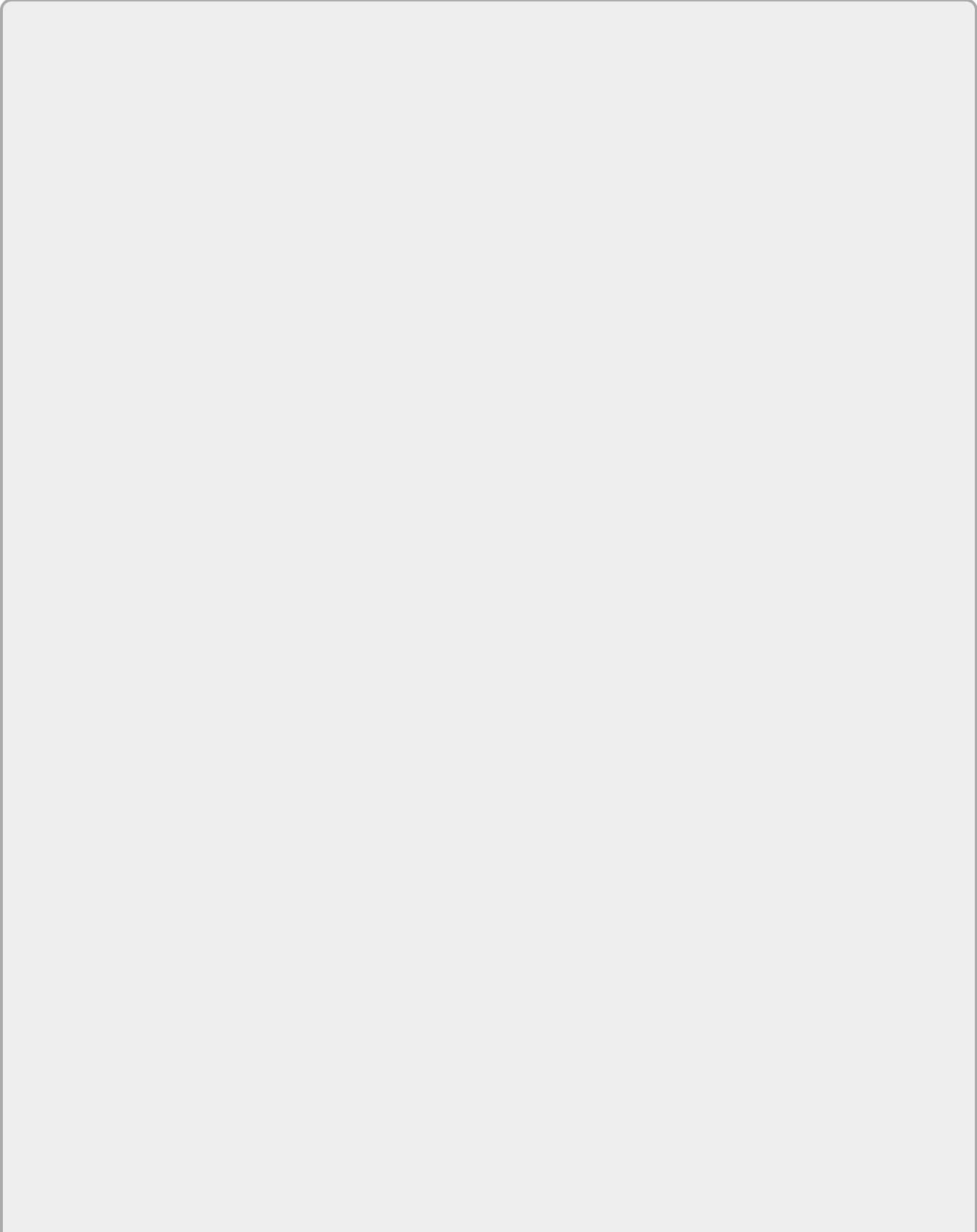
Results

The open core business model of DNN Corp. allowed it to create a viable commercial business. In the years from 2009 to 2014, it achieved significant revenue growth, expanded its operations and headcount, benefitted from investments and partnerships with a variety of highly respected technology companies, and received many industry accolades. These included recognition in the 2011 *Inc. 500* Top 20 placement in the Fastest Growing Company list for British Columbia for five straight years and representation on a variety of analyst reports, including the Gartner Magic Quadrant. The commercial traction allowed it to invest in the community, enhancing the open source platform, providing valuable community services, and creating opportunities for members of the ecosystem to meet and interact with one another. As long as the company continues to operate as a wise and generous steward of the open source project and community, operating with intellectual honesty and integrity, there is no reason why the goose will not continue to bestow it with golden eggs for many years to come.

SUMMARY

DNN's commercial success is a result of advanced technology, disruptive market conditions, and a superior distribution model. The viral growth of the open source project attracted the right combination of users and vendors and created a powerful commercial ecosystem. This created a viable commercial business opportunity for DNN Corp.

Chapter 20
Evoq Content



What You Will Learn In This Chapter

- Discovering content creation tools
- Understanding permissions, workflow, and versioning
- Uncovering Evoq Content's optimization features
- Discovering integrations support

Evoq Content is a powerful, easy-to-use, and extensible Content Management System (CMS). For most organizations, the website is the center of their marketing universe. Evoq Content empowers marketers to easily and quickly bring a website to life with captivating content that attracts and engages prospects and customers.

Evoq Content is tailor-made for today's modern marketer. Its drag-and-drop user interface makes it easy to create content, while its integration with cloud storage and marketing automation systems saves marketers valuable time.

At the same time, IT managers and developers can take advantage of the extensibility of the solution through its APIs and third-party extensions.

Evoq Content is a commercial product built on top of the DNN Platform, and it includes essential features for today's digital marketer. These features include real-time personalization, on-page analytics, third party integrations, and more.

This chapter covers some of the main features found in Evoq Content.

Content Creation

Today's digital marketers need to manage much more than their organization's websites. They work with colleagues, PR agencies, design agencies, and freelancers, sharing documents and files across the Web. They connect their websites to other systems, such as CRM and marketing automation. All while ensuring that visitors find the information they need.

Evoq Content makes life easier for today's digital marketer by providing content-creation capabilities that go far beyond content management.

Asset Manager

Evoq Content provides centralized access to cloud-hosted content repositories (see [Figure 20.1](#)). It includes built-in connectors to Box, Dropbox, Amazon S3, Azure Storage, and SharePoint, making it easy to access any piece of content or digital asset from a single window. Marketers can conveniently share documents with third parties without the need to contact IT to upload large files.

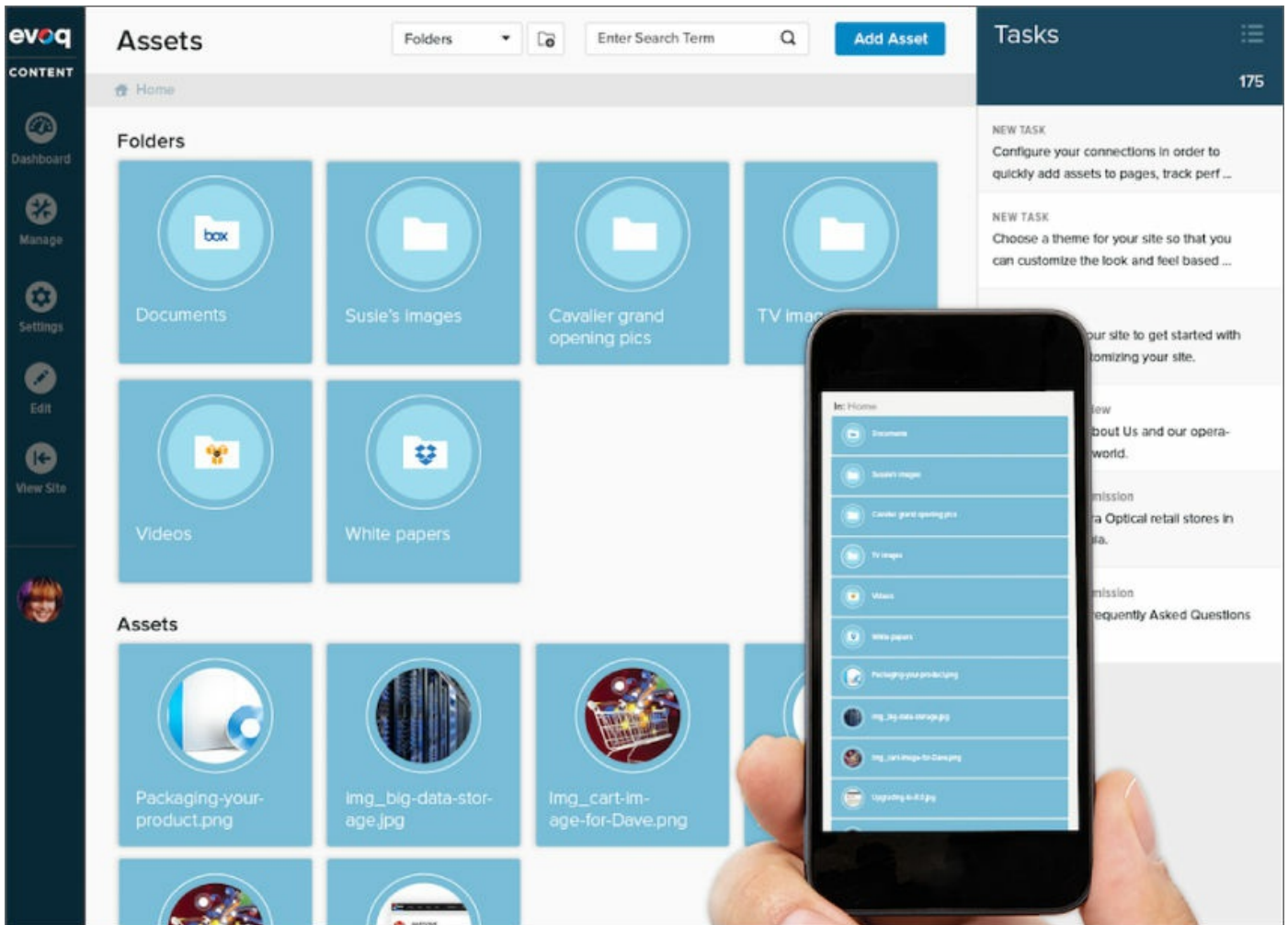


Figure 20.1

The Asset Manager allows marketers and content producers to use preferred content storage services or use multiple services without moving files, duplicating effort, or wasting valuable time.

Flexible Content Layouts

Evoq Content provides a convenient collection of built-in layouts that you can apply to pages (see [Figure 20.2](#)). In addition, you can assemble your own via a simple-to-use drag-and-drop interface. If you create a custom layout, you can save the new layout as a template, making it available for reuse on other pages. This feature enables marketers to build pages and edit content more quickly and efficiently, reducing duplicate effort and rework.

Template Management

[← Return to Page Management](#)

[View Recycle Bin](#)

New Template



2 Column



3 Column



Single Column



4 Column



5 Column



Avera Single Column

Figure 20.2

Permissions, Workflow, and Versioning

Permissions and workflow can be used together to manage a growing team of content managers. As more people contribute content to the site, you'll be able to instrument reviews and approvals, as well as monitor and enforce consistency with brand standards. Evoq Content's granular permissions feature allows you to dictate the actions enabled by particular groups of users.

Permissions

In small organizations, website management is typically assigned to one or two staff members who produce, review, approve, post, amplify, and monitor the content. But in larger organizations, the structures are more complex:

- Regular users generate text, visuals, or finished pages for specific sections of a site.
- Admins approve content or send it back for revisions before allowing it to go live.
- Managers work to repurpose or repackage content on social media and monitor its impact.

Evoq Content allows you to define granular, role-based permissions that give users access to some site sections and functions, but not others. You might have different teams in your organization, from marketing to engineering to sales, so it may be essential to set up granular permissions that reflect the duties assigned to different role-based personas.

Compared to DNN Platform, Evoq Content adds a wider range of highly granular permission options. As your organization grows, you'll have to distribute the responsibility for updating content across your team for different sections of your site, using fine-grained permissions to prevent unauthorized access to individual pages or modules.

This process puts managers in charge of the sections that correspond with their actual duties, while preventing teams from accidentally publishing changes or pages that fall outside their jurisdictions. That process puts you in control of your online content, while saving a lot of time for marketing and IT, the departments that would otherwise have to manage an entire site.

Workflow

Professional websites with multiple content editors are complex to manage. You need to make sure that the message, tone, quality, brand, and so on, of your overall website are consistent. To solve this challenge, Evoq Content offers a Content Approval Workflow that allows you to review and approve content at different stages of the process so that your visitors only get to see the content that reflects your brand and message.

Evoq Content offers three workflows out of the box: direct publish, save draft, and content approval. The three workflows differ on the number of stages that your content has to go through before it goes live in your website. In addition, you can create custom workflows and give approval permissions for each step of the workflow.

AutoSave

How many times have you accidentally closed a page or a document without saving your work, or the power goes out before you're able to click Save? Evoq Content includes an AutoSave feature that ensures the content you've created or modified in the editor is stored in a temporary location. This means that the content survives, regardless of external events such as a lost Internet connection or user error.

After an error occurs, you can still retrieve your changes. Just return to your editor and you'll be prompted to retrieve unsaved changes.

Content Versioning

Evoq Content's versioning is a feature that allows you to track the history of content for later re-collection, retrieval, or confirmation of events. Every time a new version of the content is approved and published, a version number is inserted at the top of the list. Even when a rollback is done, a new version is inserted so that the history between the latest version and the version rolled back to is not lost.

This feature enables you to “roll back” to prior versions of a page, if needed. For example, if an erroneous pricing update was applied to a page, you can roll back to the prior version to preserve the original pricing information.

Version Compare

Having the ability to keep the history of a piece of content is a very valuable feature, but you often find yourself wondering what exactly changed from one

version to the next or from an old version to the latest one.

With Evoq Content, you can select any two versions of your content and see them side by side. The system highlights the changes that occurred between one version and the other.

Optimization

It's no longer good enough to create web content for all site visitors. When customers visit a website, they expect to find the information they need immediately. Customers who don't find what they need leave quickly, and you'll have diminishing chances of having them return.

Evoq Content provides a number of optimization features that help marketers create productive and delightful experiences for their visitors.

Real-Time Personalization

Today's web visitors expect personalized experiences. In other words, they need to see what they want, when they want it. Evoq Content enables you to build personalization rules that tailor site content based on the visitor's profile (see [Figure 20.3](#)).

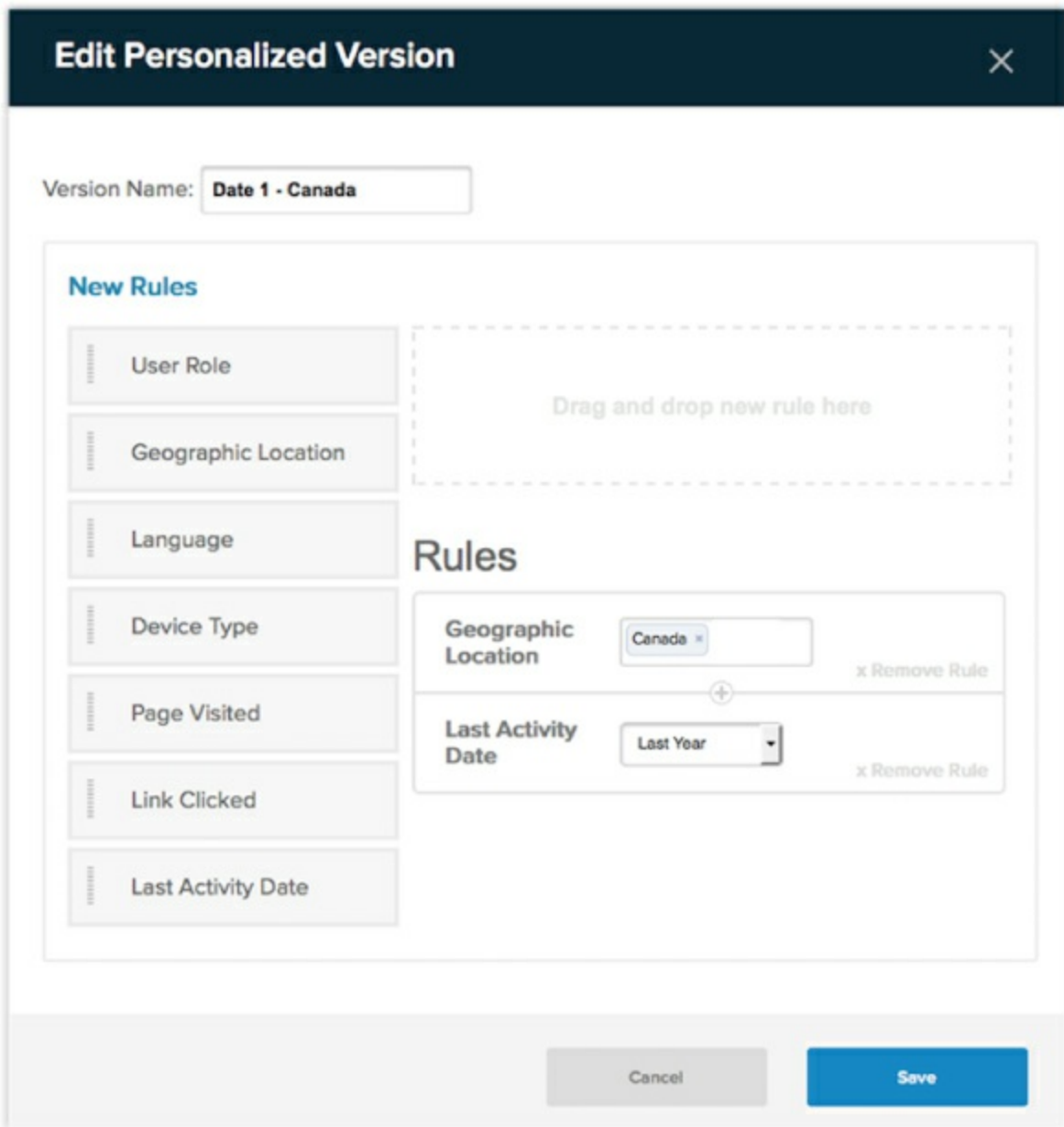


Figure 20.3

Rules are based on user role, geographic location, language, device type, page visited, link clicked, and last activity date. Personalizing content to the visitor's individual profile has been proven to increase conversions and customer engagement.

Analytics

Evoq Content's Analytics feature enables marketers to quickly see how their content is performing without needing a high level of proficiency with analytics or access to a separate web analytics system. Evoq Content provides statistics on page views, referrers, unique visitors, and conversions (see

Figure 20.4).

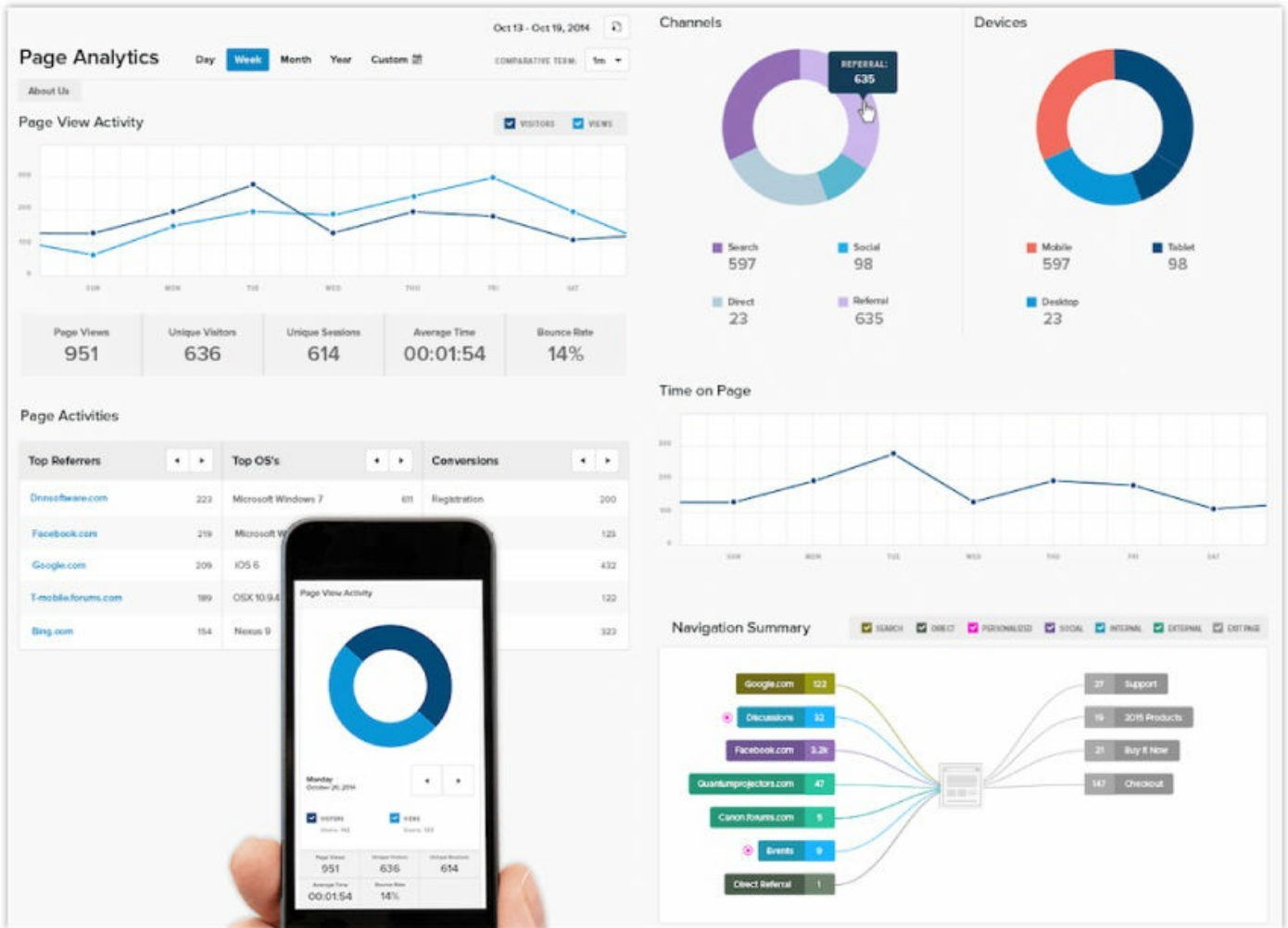


Figure 20.4

Analytics are presented as an overlay on each page, via data collected, and managed by Evoq Content. Marketers can also view a navigation summary to see how visitors are coming to a page and how they are leaving. Analytics helps you quickly understand the performance of each page of your site without having to log in to a separate web analytics tool.

Integrations

In a 2014 research report, DNN discovered that 53% of marketers at mid-sized companies use five or more marketing technology solutions, while 15% use 10 or more marketing solutions. In short, marketers rely on solutions from numerous vendors, and many of these systems need to integrate with one another.

Evoq Content simplifies the life of today's digital marketer by providing built-in integrations to widely deployed marketing applications. Evoq Content connects easily to these applications, so you can use familiar and preferred solutions to extend your website's functionality:

- **Salesforce.** Publish content to your Salesforce instance based on stage of the buying cycle and measure its impact.
- **Marketo.** Integrate Marketo easily without programming knowledge. Add the Marketo Munchkin code and deploy Forms 2.0 in just a few keystrokes.

Summary

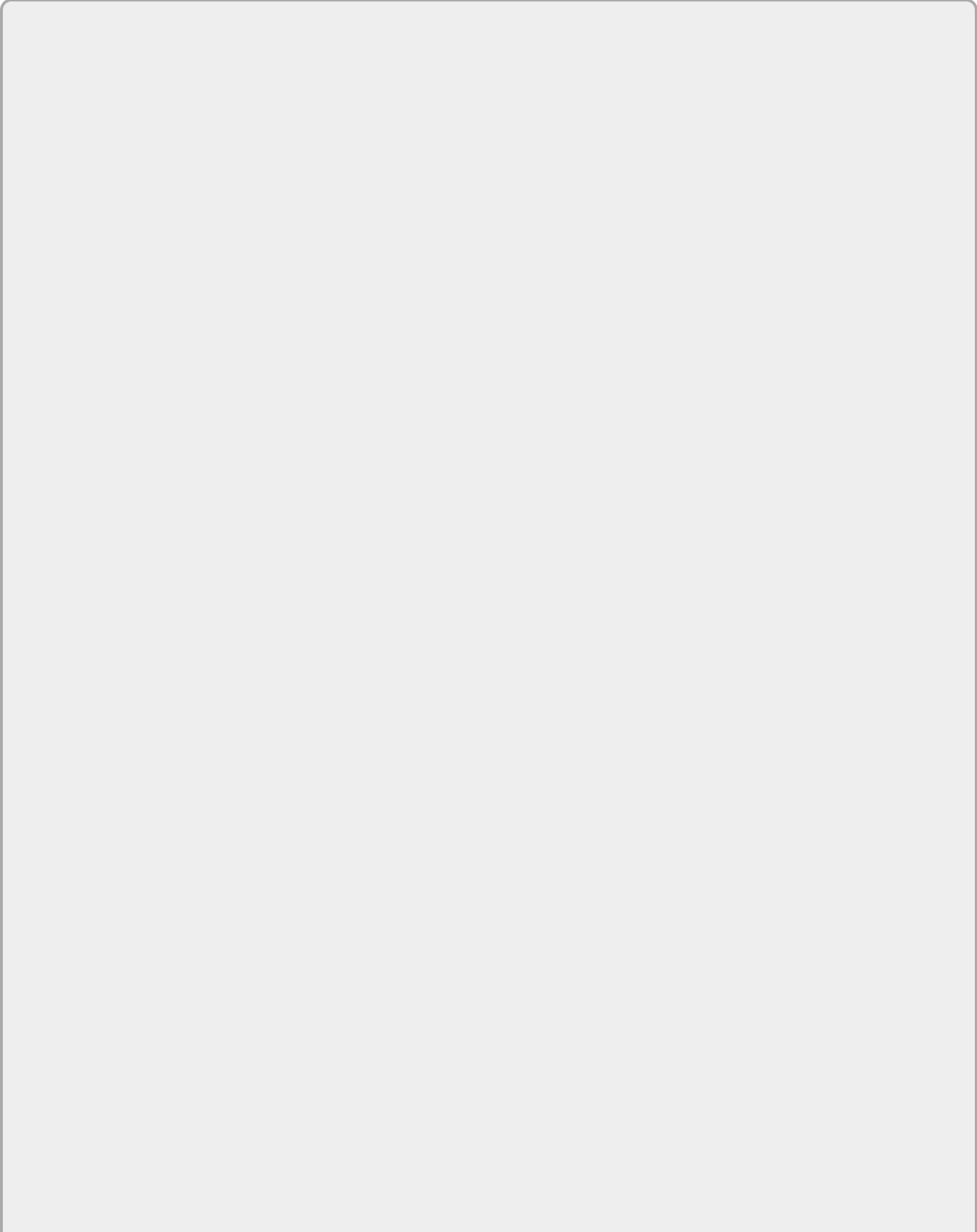
Evoq Content is a very powerful, scalable, and high-performance web content management system for small, medium, and large organizations. Some of the features you'll find include the following:

- **Content Creation.** Capabilities tailor-made for today's time-constrained modern marketer. Leverage numerous out-of-the-box content layouts, or use a drag-and-drop user interface to create your own.
- **Permissions, workflow, and versioning.** Set granular permissions on your content pages and then use workflow approvals for important checks and balances on content updates.
- **Optimization.** Real-time personalization, used in conjunction with on-page analytics, enables marketers to deliver personalized web experiences to visitors while assessing (in real time) how pages are performing.
- **Integrations.** Take advantage of built-in integrations to Marketo and Salesforce, saving your IT team valuable time in connecting your website to these systems.

This chapter covered some of the most important features of one of the most popular web content management systems for small, medium, and large organizations. If you want to try it for free, you can visit

www.dnnsoftware.com/evog-free-trial.

Chapter 21
Evoq Engage



What You Will Learn In This Chapter

- Understanding the community management tools
- Discovering the community-oriented modules

Evoq Engage is an easy-to-use online community solution that includes a powerful content management system and platform. It is built on top of Evoq Content and includes all of its features for creating, editing, and managing content. Like Evoq Content, it is a commercial product and includes functionality for enabling user engagement and the tools that are necessary for managing and measuring that engagement.

Management Tools

The features of Evoq Engage can be divided into basically two types of areas based on the perspective of the website user. The management tools, the first of these two types, are used to configure and maintain an online community. The management tools are all exposed from the persona bar and are only available to users who are logged in and have appropriate permissions.

Persona Bar's Dashboard

The persona bar is a toolbar exposed to three different user types, or personas, within Evoq Engage. It serves as a sort of control panel but is visually very different from the one seen by administrators and super users. The three types of personas are community managers, content managers, and content editors. Based on the persona of the viewing user, what is displayed in the persona bar varies. In this chapter, the focus will be on the community manager persona.

For content managers and editors, the toolbar offers access to items centered on the CMS aspects of the product, such as editing content and creating new pages. It also provides a means for viewing details about page traffic (see [Figure 21.1](#)).

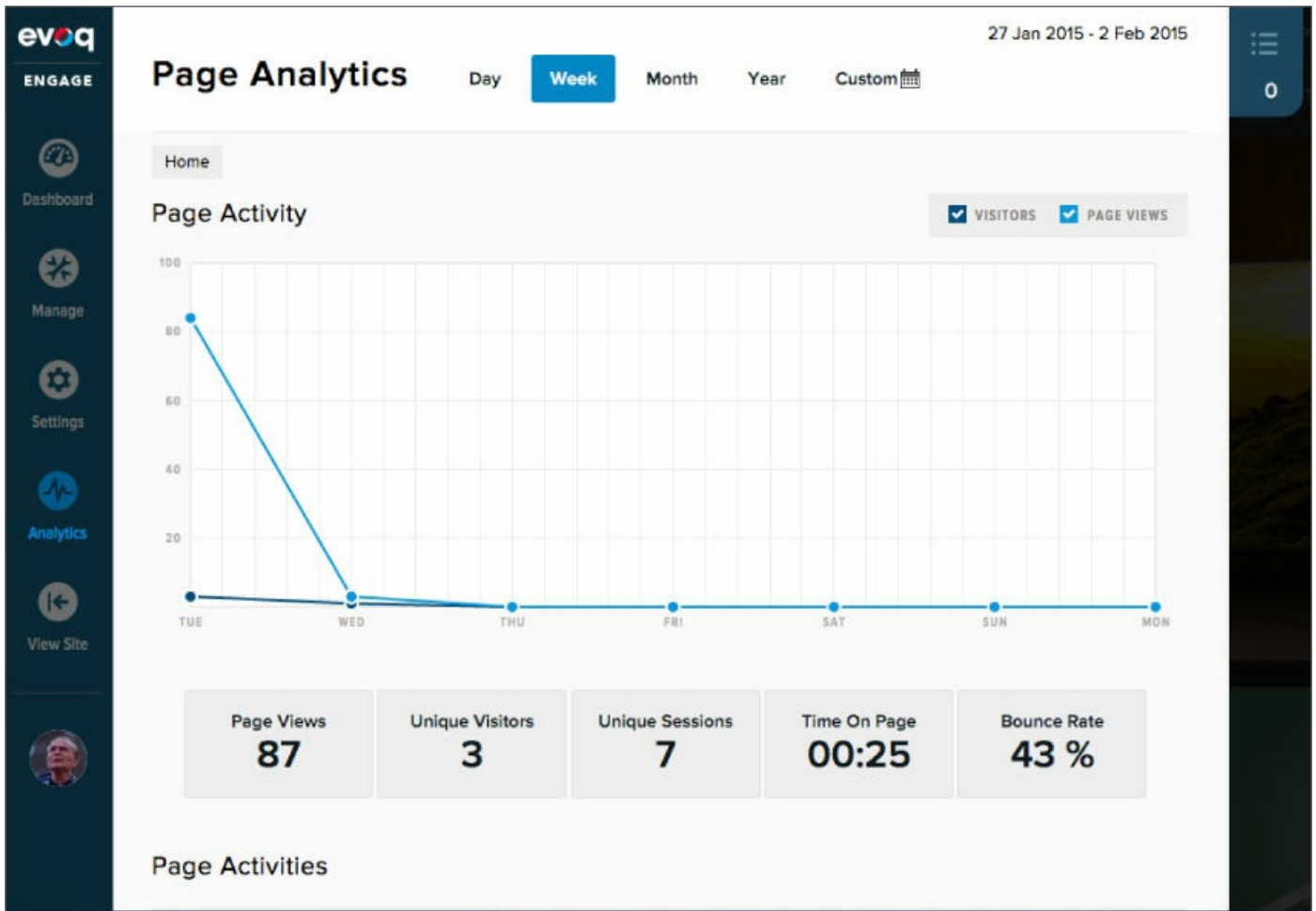


Figure 21.1

For community managers, the tools exposed in the persona bar are those that make managing an online community less of a daunting task. It is at the center of a community manager's daily activity, and clicking one of the many items located on the toolbar can access everything necessary to manage the online community.

Overview

The analytics in Evoq Engage are exposed via the dashboard icon and the submenu items within the persona bar. When clicking the Dashboard icon, community managers are presented with an overview of their community's health. Key areas about community engagement are highlighted in a series of reports displayed within the dashboard. In this initial overview, all areas of the site are reported on within a single all-encompassing view.

Below the community health section of the dashboard is the Adoption, Participation & Spectators chart (see [Figure 21.2](#)). Adopters are users who

have registered on the site within the selected period. Participants are the community members who have done something that contributes to the community such as authoring, editing, subscribing, or interacting with content. Spectators are community members who have viewed content only and not interacted with it.

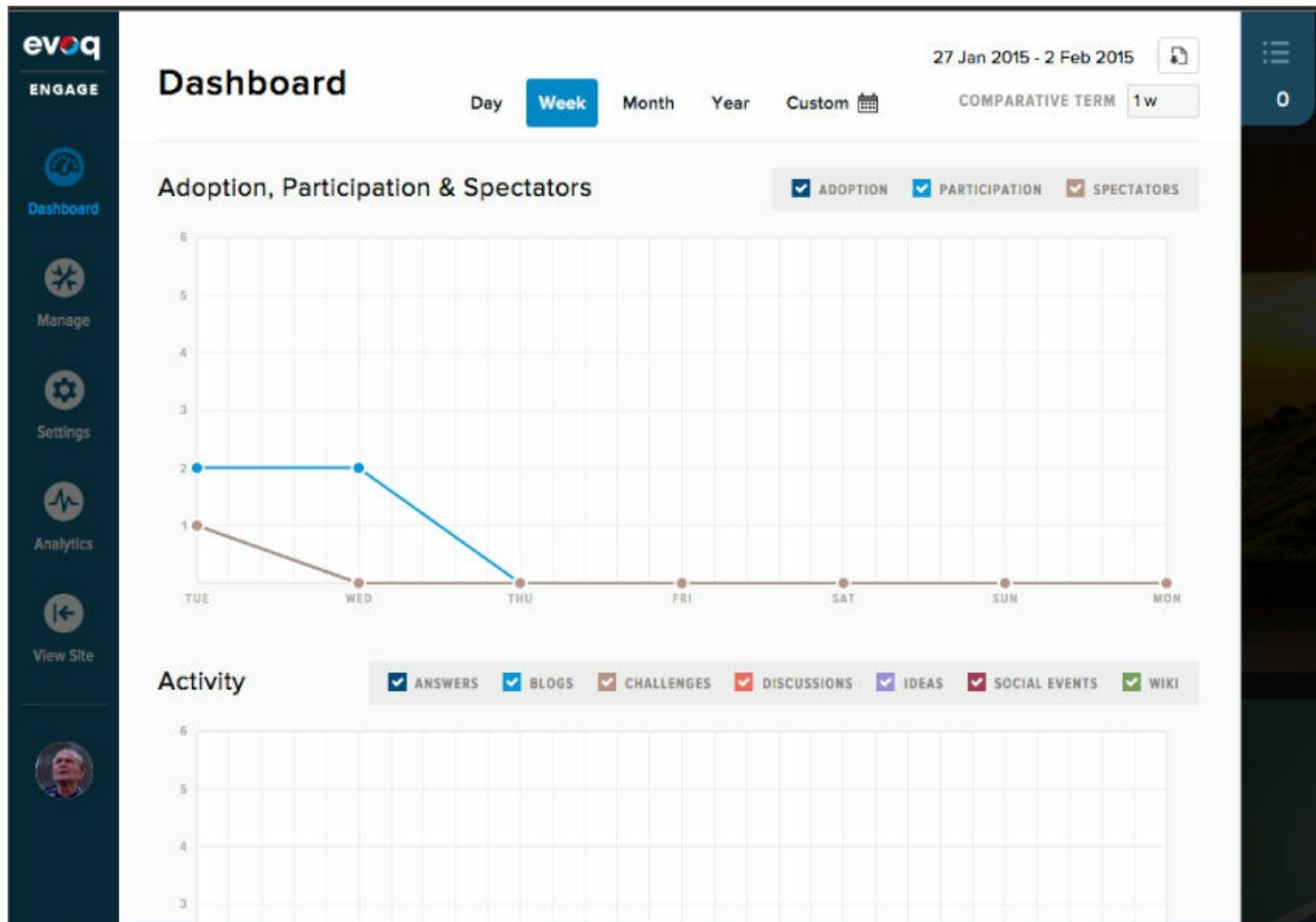


Figure 21.2

Under the Adoption, Participation & Spectators chart is the Activity chart. This chart breaks down how much activity each community module has had within the selected time period. Beneath this section is a breakdown of the trending tags used throughout the community as well as which content is popular. Both trending tags and popular content are determined based on how much engagement the content has had over the selected period. The amount of time on a page as well as the top community users are also shown within the dashboard.

Modules

Hovering over the dashboard icon in the persona bar exposes a submenu that lists all the community modules in Evoq Engage. Everything discussed previously, besides the Adoption, Participation & Spectators chart can also be filtered down to the individual community module. This allows a further breakdown of who is participating and where within the Evoq Engage community.

Persona Bar's Manage Section

The Manage section of the persona bar is centered around things that require attention more often than settings but are not necessarily something a community manager will use on a day-to-day basis.

Assets

The Assets section of the persona bar is available from the Manage icon (see [Figure 21.3](#)). This area is a way for the community manager to manage all files on the website. It not only displays the files on the site but also displays those from cloud providers like Box and Dropbox.

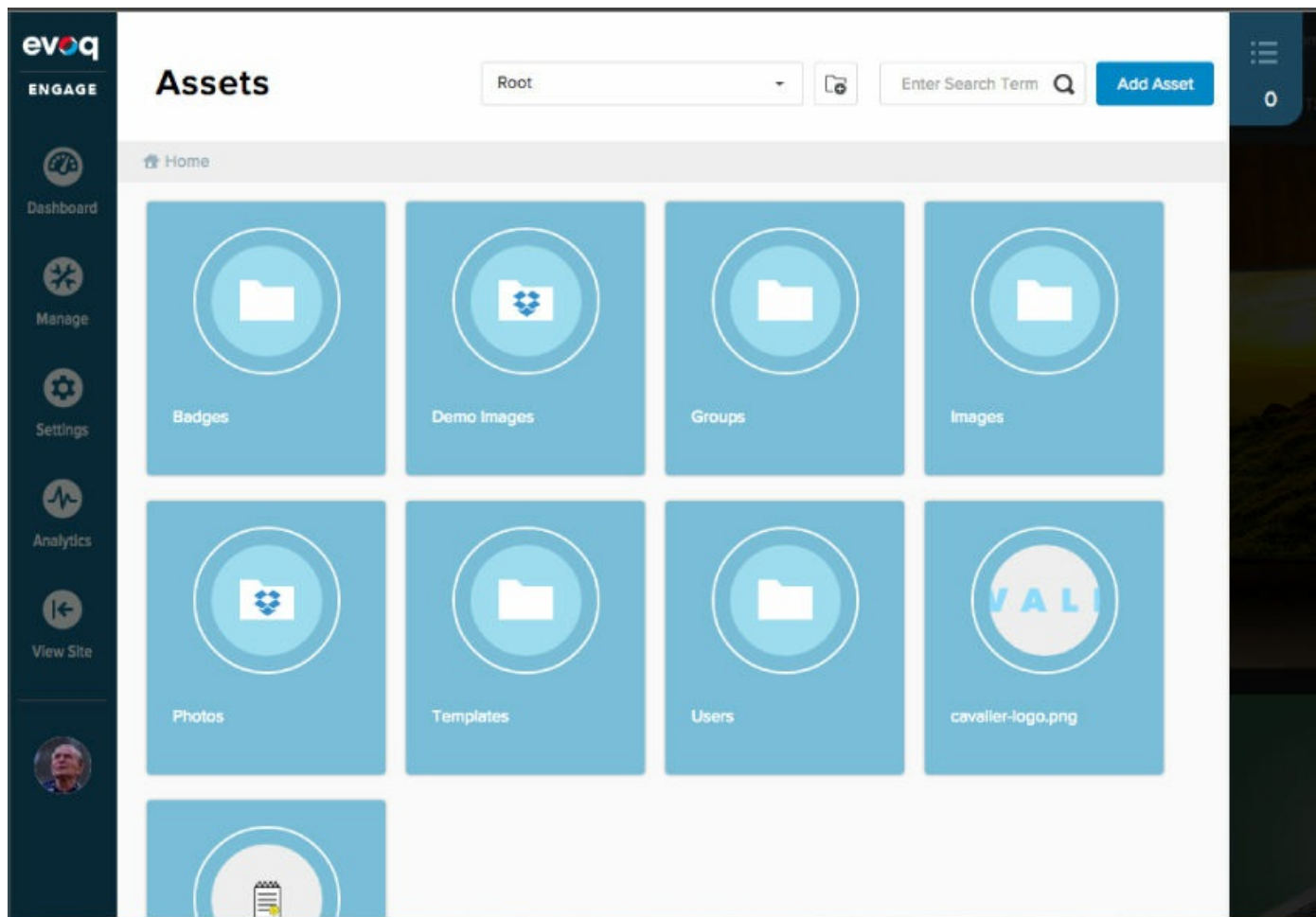


Figure 21.3

Clicking any folder opens the folder and shows you the other folders and files that it contains. You can edit the title of the files and folders and also get direct links to them.

Users

The Users section is also a submenu item of the Manage icon of the persona bar. By default, it lists the latest community members at the top, with the oldest member being the very last one in the list. This is an alternative for the User Accounts module under the Admin menu.

Users can be searched by typing the username, display name, or email address for which you are searching. The results are filtered immediately in the list below. Clicking any user expands the user to show additional details about them such as their reputation and influence as well as the areas in which they were last active (see [Figure 21.4](#)). You can also navigate to a user's assets from here to delete any items they may have uploaded that are not meant for community consumption.

The screenshot shows the 'Users' page in the Evoq Engage interface. At the top left is the 'evoq ENGAGE' logo. A vertical sidebar on the left contains navigation icons for Dashboard, Manage, Settings, Analytics, and View Site. The main content area is titled 'Users' and features a 'Find a User' search bar with a 'Filter by name' input and a 'Create User' button. Below the search bar is a table with columns for NAME, EMAIL, and JOINED. The first row is highlighted in blue and shows a user profile for 'Content Editor' (username 'ce') who joined on 1/27/2015. Below the table, the user's profile is detailed with a 'Content Editor | ce' header. On the left, there are three statistics: RANK (8), REPUTATION (6), and EXPERIENCE (8). Below these are 'EDIT POINTS', 'ENGAGEMENT' (0), and 'INFLUENCE' (0). At the bottom left, a box indicates '9 TOTAL CONTENT CONTRIBUTIONS'. On the right, the 'RECENT ACTIVITY' section lists four entries: 'Text/HTML Module Created 7 days ago' and 'File added 7 days ago', both associated with 'EVOQ CONTENT LIBRARY'. The 'LAST ACTIVE' status is shown as '7 days ago'.

Figure 21.4

Persona Bar's Settings

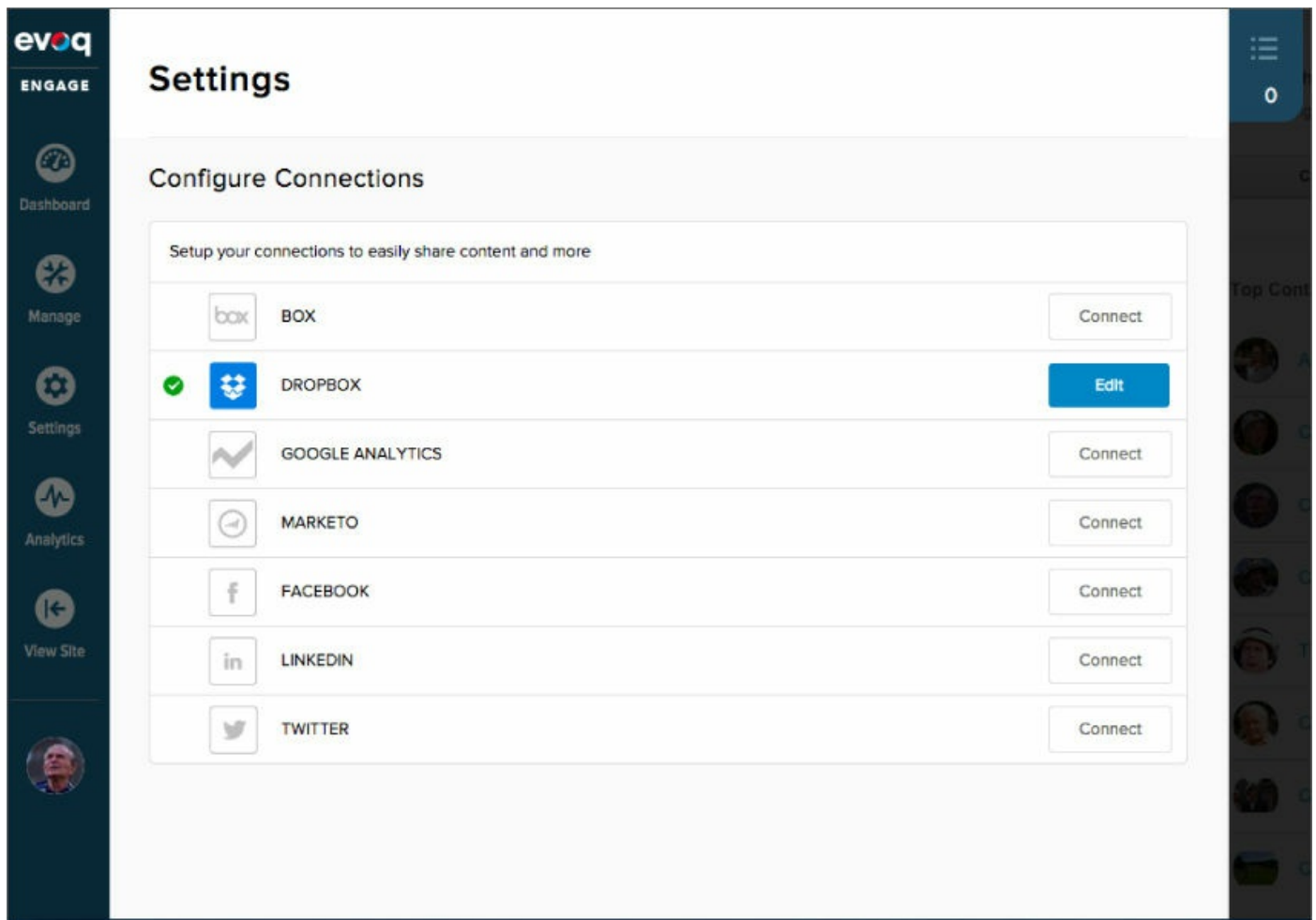
Unlike the items under Manage, items under the Settings area are generally set up once and are rarely revisited by the community manager.

Connectors

Connectors are a way to connect your online community with outside services. To manage connectors, the community manager must select the Connectors submenu item under the Settings icon of the persona bar. Each connector has its own set of fields but usually requires some type of unique identifier. For configuring the Marketo connector, for example, you simply set a unique ID and save the settings. To obtain this ID, you must have an account already set up with the provider of the connector you are setting up.

Evoq Engage includes multiple connectors and continues to add to the list. At the time of the publishing of this book, the following connectors are

available: Box, Dropbox, Google Analytics, Marketo, Facebook, LinkedIn, and Twitter with Amazon S3 and Azure on the way (see [Figure 21.5](#)).



[Figure 21.5](#)

Gaming Mechanics

The gaming mechanics within Evoq Engage contain both gaming and reputation aspects. Both aspects rely on a points system, which can be broken down into two distinct types: experience and reputation. Experience Points (XP) only increase over time and provide insight into how a community member has contributed to the online community over a lifetime of activity. XP is only displayed to community managers in Evoq Engage, and community members never see this score.

Reputation points, on the other hand, can go up and down over time based on how the community member contributes and interacts within the community. This reputation score is what is displayed everywhere within the Evoq Engage community to represent the community members score. The

reputation points are also utilized for privileges, which enable modules to expose functionality based on an end user's reputation score. The required reputation score to gain a privilege is controlled by the community manager.

Community modules generally make use of both scoring actions and privileges, and all aspects associated with content authoring and editing within the CMS product can also be associated with XP and reputation points. It is worth noting that any third-party modules integrated with the mechanics engine can also have their reputation and experience points managed from the persona bar's gaming settings area as well.

Scoring actions also allow community managers to create badges. Badges provide community managers with a way to give community members an award for doing a series of actions on their website. Each badge can have its own image and can be associated with one or more scoring actions. You can require end users to complete the action one or multiple times, and you can require them to perform these actions within a specified time frame. Badges can be managed from the same area as points and privileges, as shown in [Figure 21.6](#).

The screenshot displays the 'Gaming' section of the Evoq Engage interface, specifically the 'Badges' management page. The page includes a search filter, a 'Create a Badge' button, and a table of existing badges. The table columns are: BADGE, TIER, TIMEFRAME, GOALS, and ACTIONS. The 'ACTIONS' column contains a 'SHOW ALL' link for each badge.

BADGE	TIER	TIMEFRAME	GOALS	ACTIONS
Detective	Gold	30	Answers	SHOW ALL
Getting started	Bronze	30	Evoq Content Library	SHOW ALL
Great Question	Silver	30	Answers	SHOW ALL
Guru	Silver	30	Answers	SHOW ALL
Oracle	Gold	30	Answers	SHOW ALL
Team Player	Bronze	30	Group Directory	SHOW ALL
Thumbs Down	Bronze	30	Answers	SHOW ALL
Thumbs Up	Bronze	30	Ideas	SHOW ALL
Up to the Challenge	Bronze	0	Challenges	SHOW ALL
Wikichievement	Gold	30	Wikid	SHOW ALL

Showing 10 items

Figure 21.6

General Settings

The Settings area within the persona bar consists of configuration items that are set up by the community manager once and are generally not visited and changed too often once the community is up and running. Influence, which allows the community manager to determine how community members influence scores, is calculated and based on a number of factors that can be set from this section.

Below the influence settings section is the miscellaneous community settings area that provides a means to control which content and comments, if any, are moderated prior to public display. Other community-oriented settings such as profanity filtering and the enabling and disabling of the WYSIWYG are also configured from here.

Tasks

Unlike other areas within the persona bar discussed so far, the Tasks area is always available regardless of the selected menu or submenu item. Tasks are displayed on the right side of the expanded Persona Bar and can be collapsed when the end user needs them out of the way. The total number of open tasks is displayed, and the oldest tasks are always listed at the top. This is done to help ensure the community manager takes action on those tasks that have been in the queue the longest, thus preventing any task from going unattended.

Simply put, the Tasks area displays a list of action items for the community manager. These tasks vary from content and users pending moderation approval to notifications of important events related to community activity. Community managers can take action on all tasks from directly within the persona bar; thus, they don't have to go from page to page across the site.

Community Modules

Users who are typical site visitors or community members will see features and functionality covered in this section of the chapter. All of the functionality discussed shares several key similarities.

For starters, the functionality is designed to take in content from community members. This content can be moderated and requires approval from assigned moderators prior to public display, or it can be set up for instant approval and public display. It is also designed to have a consistent look and feel. This consistency and the intuitive user interface lower the barrier for community members to contribute, which results in a more pleasing experience and leads to more participation by community members.

It is important to note that not all communities require all of the tools and functionality discussed in this chapter. All of these tools are included to cover various community types from company intranets to public-facing branded communities for organizations of all sizes. The primary objective when deciding what to implement in an online community is determining what features give community members the most value and keep them engaged.

A second key similarity is that each member-generated content module has moderation capabilities assignable per module instance. This allows community managers to identify moderators for each specific member-generated content module, thus dividing responsibility for community management. Dividing responsibility among the various areas within an online community, which is more important in larger communities, helps ensure all areas of the community are monitored and maintained.

Another key similarity in each module is they are tightly integrated with the powerful search engine included in the DNN platform. All member-generated content is indexed and available in site search results. The search engine indexing also takes the permissions associated with this content into consideration when returning search results. This ensures search results only display content available to the user who performed the search—a very important aspect for communities that have a need for content that isn't available to all visitors.

A final key similarity in the member-generated content modules is that they are all integrated with the core social elements in Evoq Engage. These core social elements, like the Activity Stream and the gaming mechanics, are

tightly integrated across all community modules and the management tools discussed earlier in this chapter.

Answers

The Answers module provides a way for community members to ask questions and get answers to those questions (see [Figure 21.7](#)). Providing a way to ask questions on a community site enables other community members or company employees to answer those questions without the need for a phone call or an email. For a business, this can reduce support costs and improve customer satisfaction.

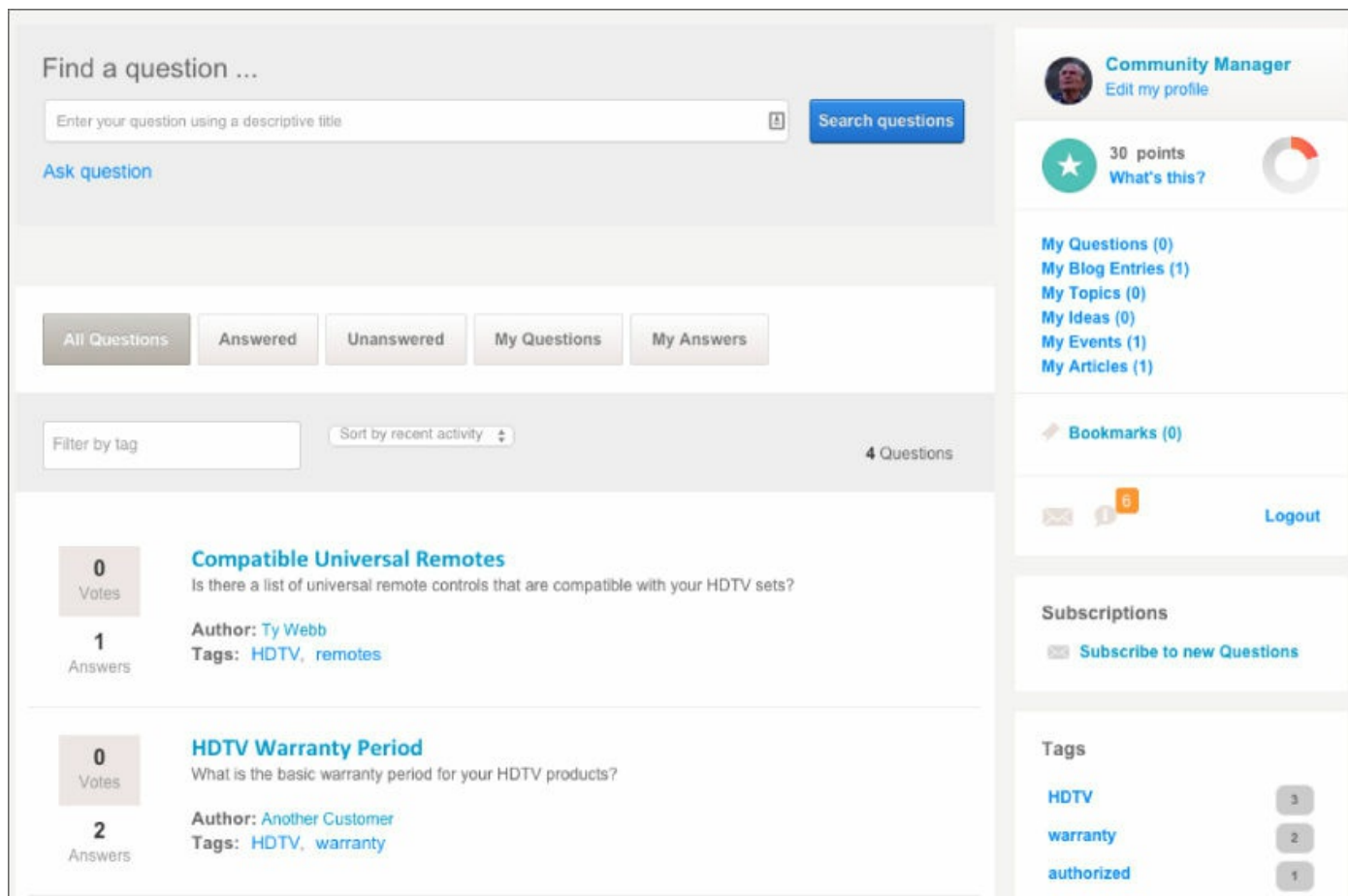


Figure 21.7

To help website visitors find answers to their questions, the Answers module allows authors of questions to accept one of the available answers as the correct one for their question. When an answer is accepted, it is always displayed at the top of the list directly below the question. This provides a means for others to easily identify correct answers to questions without having to read through all of the available answers.

The Answers module allows community members to vote on answers provided by other users. This is another way that the community members can help one another, thus keeping them engaged, as well as providing more value to all members. Answers with higher scores rise to the top, sitting just below the accepted answer, if available.

Blogs

The Blog module allows users to generate article-like content that can be viewed and commented on by other community members. The Blog module can be configured, via the module's permissions, to control which users can author content within the module. Users who are granted the Blogger permission can contribute blog content.

Challenges

The Challenges module was designed with community managers in mind and provides them with a way to empower the advocates of their community. Community managers create challenges to accomplish various objectives that benefit their community, and as they do so, they choose who is permitted to participate in the challenge.

Challenges can be divided into one of two general types: internal and social network. Internal challenges are used to accomplish tasks within the online community. For example, get a specific question from the Answers area answered or get some members commenting on a specific blog article.

Social network challenges are used to increase a community's reach beyond the website and current members. It does this by integrating with several of the social networks: Facebook, LinkedIn, and Twitter. Social network challenges allow community members to share content on these networks and rewards them for it via reputation points and badges, which were discussed in the “Gaming Mechanics” section of this chapter.

Discussions

The Discussions module is an area that permits community members to have online discussions with each other in a fashion similar to forums. Unlike a traditional forum, where subforums are used to group topics into predefined categories, the Discussions module in Evoq Engage uses a flat structure where discussions are categorized using free-form tags entered by the content

authors. The flat structure minimizes the number of clicks used to find content in specific areas.

Discussions are more appropriate than the Answers module for member-generated content that usually has no definitive answers. For example, a question asking users their favorite color is much more appropriate for the Discussions area than the Answers area.

Ideas

The Ideas module provides a way for community members to contribute their own ideas and also provides the means for the community to vote and comment on them. Community managers can set the maximum number of votes permitted for users on all ideas, as well as the maximum number of votes that can be contributed to any single idea. Limiting the number of votes forces the community members to make difficult choices and thus helps community managers prioritize what ideas are valuable to their community.

Events

The Events module was designed to enable location-based or virtual events for community members. Location-based events can enter an address where the event will take place, and a map of the location is displayed in the event details to members.

Event organizers can limit the number of attendees for cases where live events have a limited number of spots available. It is also set up for users to be able to reply to invitations with an attend, maybe, or decline status.

Wiki

The Wiki module allows community members or a company's employees a way to create online living documentation that constantly evolves. It enables WYSIWYG HTML content authoring and utilizes that markup to produce a table of contents automatically so users viewing content can quickly jump to specific sections.

Others

There are several community modules that aren't geared for user-generated content. Instead, these modules focus on very specific pieces of functionality, and there isn't much depth to them. They have a similar look and feel the

modules previously discussed, but most of them are not taking in content from the end user.

Activities

This module is used to show all users what actions they can take to earn reputation points on the site. Items that have no value or are negative are not displayed here. The positive scoring items are the only ones displayed because many communities may not use a lot of the features. In those situations, it is best to set the reputation score to zero for all those items so they are not displayed within this module.

Activity Stream

The Activity Stream module is an aggregated view of activity in a community. It surfaces all member-generated content in a centralized interface that can be filtered by area. As community members engage with items in the activity stream, they are auto-subscribed to notifications on updates related to the content with which they have engaged. For example, a user who comments on anything in the activity stream will be notified via email any time someone else comments on the item (see [Figure 21.8](#)).

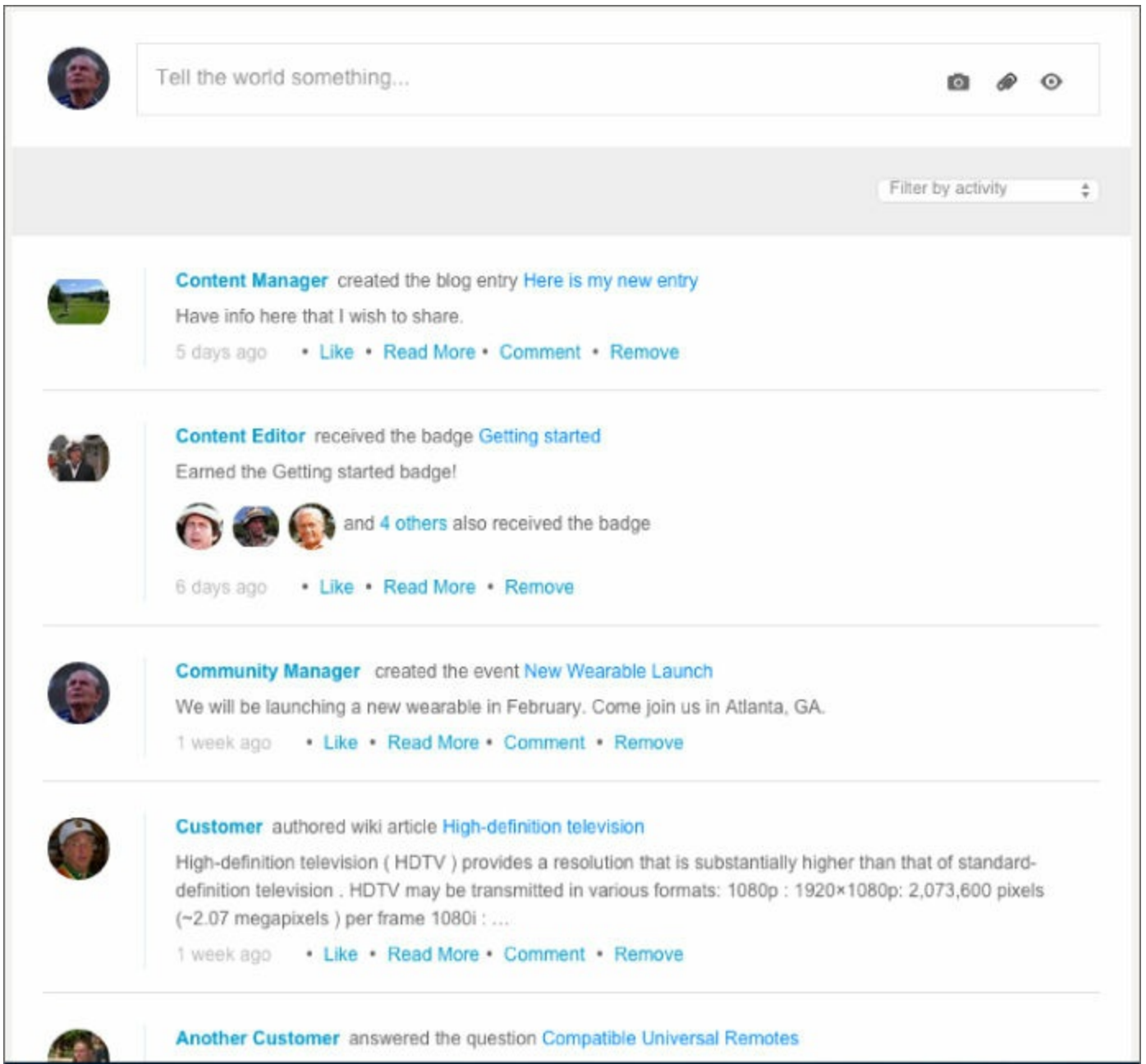
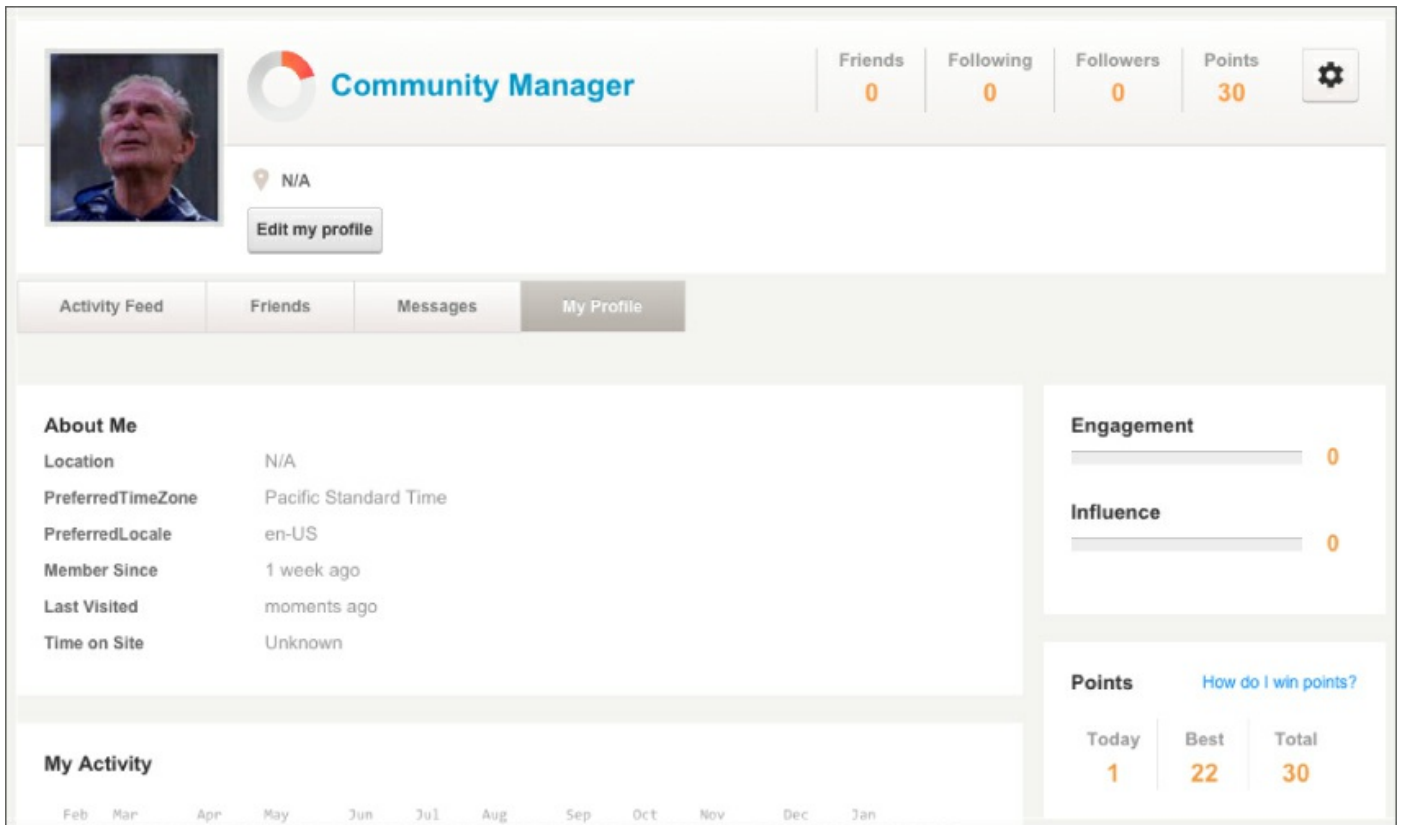


Figure 21.8

Member Profile

The Member Profile feature contains a series of modules that display various aspects of a user's profile to community members (see [Figure 21.9](#)). A few things shown include a community member's profile details, community activity, reputation score, and badges. It takes into account all of the privacy settings, so it only shows things the community member permits to be displayed to the rest of the community.



[Figure 21.9](#)

Leaderboard

The Leaderboard is a module that displays a list of community members sorted based on the member's reputation score. It can be configured to show this list based on one of several predefined intervals: last day, last week, last month, last year, or lifetime. This option is important because as your community grows you may want to implement multiple leaderboards in the community that are configured differently to help keep your community engaged.

The points displayed are reputation points. The number of reputation points within the selected time frame is what determines the rank of the users with the highest score in the result set being at the top of the list.

Group Spaces

The Group Spaces feature consists of a group directory module and a group spaces set of modules. The Group Directory module displays a searchable list of social groups within the community. Depending on the configuration, community members can create their own social groups.

User Badges

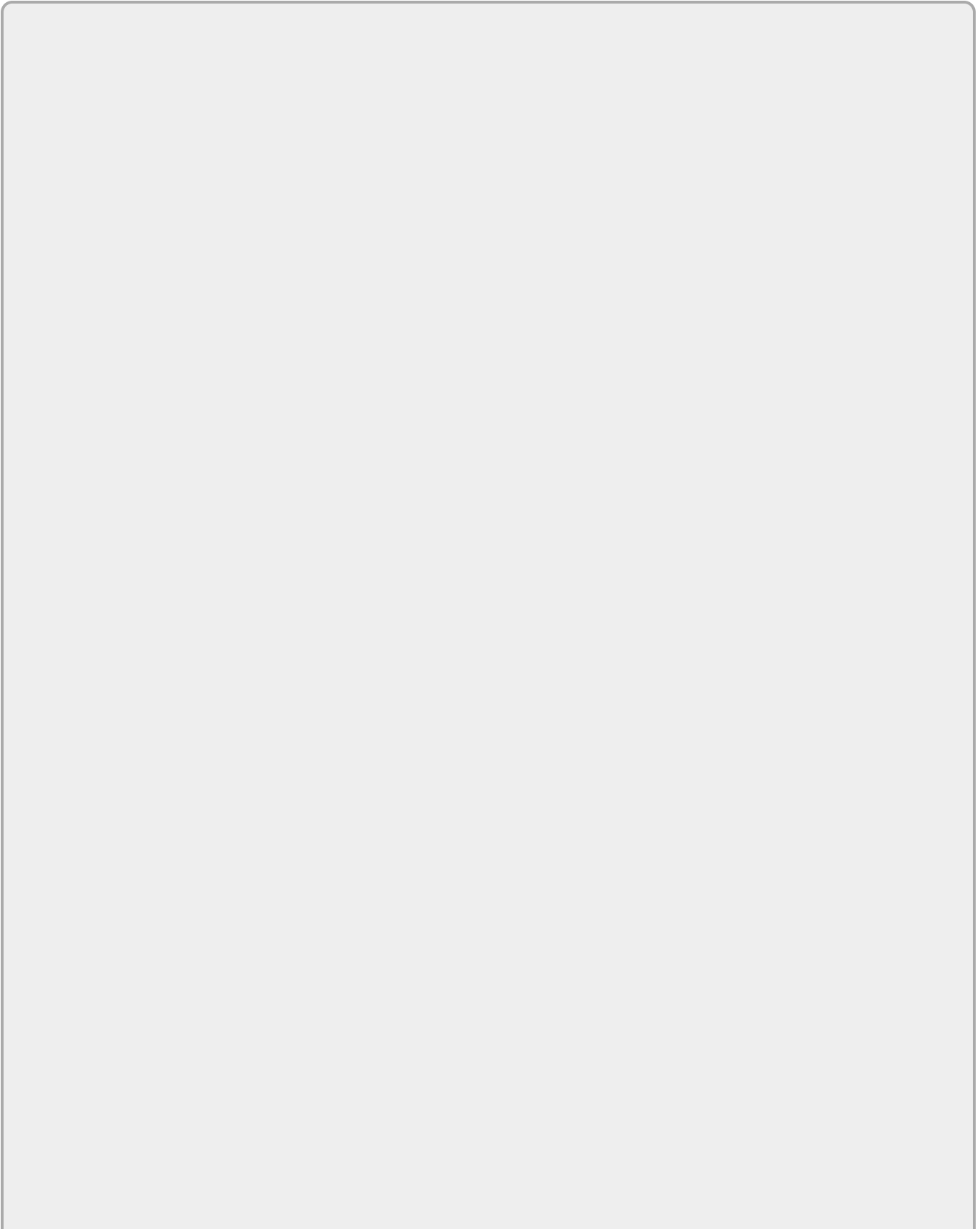
The User Badges module displays a list of all the available badges within the community. If the end user hovers over a specific badge more information about the badge, such as how to earn it, is displayed. It also shows the last three users who earned the badge providing more recognition for them that is visible to others.

Summary

This chapter covered the features included in Evoq Engage that are classified as either management tools or community modules. Management tools allow the community manager to configure, monitor, and run the community from day to day. The community modules are what are used for the community members to engage within the community.

Chapter 22

The DNN Store



What You Will Learn In This Chapter

- Utilizing the store
- Making money with the referral program
- Selling products on the store

The DNN Store is an online marketplace where vendors who develop extensions for the DNN platform can sell those extensions to end users and system integrators. The store started out initially as Snowcovered.com, which was a marketplace for IBuySpy extensions and SharePoint web parts. As IBuySpy morphed into DotNetNuke, and as the DNN platform grew in popularity, so did Snowcovered.com. On August 27, 2009, DNN Corp. officially announced the acquisition of Snowcovered.com. DNN Corp. continued to operate the site until February 6, 2012, when the DotNetNuke Store was launched as a redesigned and upgraded version of Snowcovered.com. Since then, development of the store has continued to progress. DNN Corp. has remained committed to ensuring that the store is a viable marketplace for vendors and platform users.

Buying from the Store

The majority of the store users are there for one common purpose: to find extensions that meet the needs of their projects. This is typically done by searching for and then researching and comparing those products. If more info is needed, customers can obtain presales support by asking the vendors questions about the products. Many of the products also offer trial versions or demo sites. Once products are selected, they can be purchased and installed.

Locating Products

There are a few ways to locate the product that you're looking for on the store. The next five sections cover the most popular ways to locate products, which include using the search and browsing the category lists, the top themes and modules lists, the top vendor list, and the vendor profile. Details regarding the use of the product list and the result filter are also covered.

Product Search

A lot of work has gone into improving the site's search functionality from the initial Snowcovered days. Every page on the site has a search bar at the top that you can use to initiate a text-based search query. The search uses a custom algorithm that includes weighted full-text results from the product description, vendor name, and product title. The results are then ranked by this value and displayed on the search results page.

The Search Filter

Upon the initial launch of the new store the search results page also included a filter bar that gave users the ability to filter by version, product type, a single category, and a single tag. About six months after the relaunch of the store, the search filter was upgraded to support filtering by multiple categories and multiple tags. Each of the categories and tags listed in the search filter is an attribute possessed by the items in your search subset. To the right of each category or tag there is a number. That number represents how many results remain if that item is added to the search criteria. Filters can be removed by clicking the red “x” to the left of the item you want to remove.

Product List Sorting

The search results page also provides the ability to sort the results in a variety

of ways. When performing a text-based search, the default (and recommended) way to sort the results is by relevance. This allows the weighted full-text algorithm to display the most relevant results first, similar to the way that a search engine like Google ranks and displays results. The results of any search can also be filtered by any of the following:

- **Published Date:** The date on which the listing was first activated on the site.
- **Last Updated Date:** The date when the listing was last modified by the vendor.
- **Average Review:** This orders the results by the review average (Σ (product reviews) / total number of reviews).
- **Best Selling (Units – All Time):** This ranks the products based on the total number of sales that product has on the store.
- **Best Selling (Revenue – All Time):** Similar to the previous item; however, the total is based on the revenue.
- **Best Selling (Units – Last 2 Weeks):** This is the same as Best Selling (Units – All Time); however, results are limited to sales that occurred in the last two weeks.
- **Best Selling (Revenue – Last 2 Weeks):** Similar to the previous item; however, the total is based on revenue.
- **Alphabetical: A to Z or Z to A:** The results are sorted by product title. If you know the name of the product you're looking for sorting the list alphabetically is a quick way to find it in the list.
- **Price: Low to High or High to Low:** These are used to sort the results by the list price. Do you have a budget in mind? Sort by price to quickly find results that fall within that budget.

Product List Details

The product list that is used to display search results is relatively straightforward. Each product listed includes a product image or thumbnail, title, and Read More button, all of which link back to that product's detail page. It also includes the vendor's name, which links to that vendor's profile. Other information displayed for each product includes the date the product was last updated, a short description of the product, and the base price of the

product. If the product has reviews posted, there is an icon that displays on a scale of 1 to 5 the average review rating of the product and next to the total number of reviews posted for that product.

Other Ways to Shop

While the product search is by far the most popular way to locate products on the store, there are a few additional ways of locating product on the store. The next most popular area of the site is the Top Sellers lists. Currently, there are three top seller lists on the store. The first one is top modules; it is a list of the top 100 best-selling modules based on sales revenue from the last two weeks. The second list is top themes; this is a list of the top 100 best-selling themes based on sales revenue from the last two weeks. The third list is the newest, and it is the top vendors list. This list shows the top 25 highest grossing sellers on the store and ranks them in descending order by product sales for the previous week. It also shows that vendor's rank from last week and whether the vendor's rank is trending up or down.

The third most popular way to shop for products is to browse for them. There are two initial paths for this—Modules and Themes. Both are located in the primary navigation on the store. Clicking either of these items takes you to the product list page. From this point these pages work exactly like the product search discussed earlier, only there is no search text applied. However, it's still possible to narrow your results by version, multiple categories, and tags, and the results can be sorted.

Another popular way to shop for product on the store is via the vendor profile. Most of the vendors on the store sell multiple products. Often these products complement each other both in design and functionality. Some vendors even provide discounts for purchasing multiple products or offer product bundles that include many of their top extensions. The top vendor list discussed earlier is a good initial way to find some reputable vendors on the store. Clicking any of the vendors in that list takes you to that vendor's profile. Another way to see the vendor profile is to click the vendor's name from any of the product details, search results, or product list pages. The vendor profile page includes a product list that displays all of the vendor's active products. The vendor profile also displays all the reviews that vendor has received. The reviews are a good way to hear what other shoppers have to say about that vendor and their products.

The Product Details Page

While attempting to evaluate the likelihood that a product will meet the needs of your project, the product details page is one of the most important pages in this process. This page is configured by vendors to describe their products. Currently, there are a few different sections included on this page that include the Product Details, Product Options, Discounts, Description, FAQ, License, Azure Compatibility, Tags, Categories, and Spotlight Reviews.

One of the more important areas on the page is the Product Details section. It includes the following information:

- Title
- Product icon
- Review average on a scale of 1 to 5
- Total number of product reviews
- Vendor name (which links to the vendor profile)
- Date the listing was published
- Date the vendor became a seller on the store
- Country of origin for the vendor.
- Average vendor review (This is computed from reviews placed in the last six months on all of the vendor's products.)
- Ask a question link (This is a good way to ask any presales questions that you might have regarding the product.)
- If you are logged in, there is also a link to add the product to your watch list.

Many products on the store have multiple options; for example, Standard Edition, Enterprise Edition, Source Edition, and so on. If the product has multiple options, these are shown below the product details area. It's important when deciding to purchase a product that you evaluate the difference between each option and ensure that you are purchasing the correct option that meets your needs. This area of the page also includes the Add to Cart button.

Some vendors provide discounts for their products. If a product has any current discounts, they are listed below the product options. It's important to

note that other than private coupon codes there isn't anything special you need to do to apply discounts to an order on the store. If you are eligible for a discount, it is automatically applied when you add the product to your cart. Many of the discounts rely on your previous order volume or ordered items. What that means is for these discounts to apply correctly, you will need to be logged in when you go to the cart page.

The next primary section on the Product Details page is the product description. This is a large area of text/HTML that is provided by the vendor. The content of this area is entirely up to the vendor to provide, but quite a bit of product information is often included here. This usually includes a product overview, a list of features, demo links, screen shots, compatibility information, project history, comparisons between product options, and licensing.

The vendors have the ability to turn presales help desk tickets into FAQs, which is a good way to find answers to common questions that other customers have had regarding a product. Not all products have FAQs, but if they do, they will be located under the product description.

Some products specify a license (EULA) statement. If provided, it is shown at the bottom of the page below the product FAQs section. If a license is provided, it's important to read and understand the license so that you can be sure that your intended use and expectation of the product are in line with the license provided by the vendor.

On the right side of the Product Details page are a few informative sections. The first is the Azure compatibility badge, which is used to report whether the product is compatible with cloud-hosted instances of DNN. The next section down is the product tags. These are tags created by the vendor to describe the product; clicking any of these tags takes you to a search results page that lists other products on the store that share that same tag. Below the tags are the categories. These behave like the tags, but the primary difference is the categories are created by the store admins, and vendors can pick the categories from this predefined list. Like the tags, clicking a category takes you to a search results page that displays other products that share that same category. The next section down is the spotlight reviews. These are reviews that the vendor has elected to feature on the Product Details page. If you want to see all the reviews for a product, click the review average at the top of this section or the review average in the Product Details section at the top of the page.

Making a Purchase

Once you find the product(s) you want to purchase, the next thing is to add the item(s) to your cart and then proceed through the checkout process. With the redesign of the store the checkout has been simplified to a four-step process.

Once you have a product in mind, the first step in the purchase process is to add the product to your cart. This is done from the Product Details page that was covered in the previous section. The Add to Cart button is located below the product details area and is often associated with product options. If you need an option other than the default, be sure to select it prior to clicking the Add to Cart button. Once added, you are redirected to the cart page. If you need additional items, you can also add them to your cart following the same process. If you need to purchase multiple quantities of an item, enter a number in that item's corresponding Quantity box, or you can use the up and down arrows to the right of the box to increase or decrease the quantity. If an item in your cart has a discount applied, the discount amount and percentage are shown in the Discount column. It's important to note that discounts are noncompounding, and the cart will automatically apply the discount of maximum value. Most of the discounts in the system are automatically applied based on previous order activity. To see if any of these discounts applies to this order, you need to be logged in. If you have a private or promotional coupon code, you can enter it at this time in the box above the cart total and then click the Apply Coupon button. If you have an item in your cart that you want to remove, simply click the Remove link to the right of the item. If you want to clear your entire cart, simply click the Clear Cart link at the bottom-left corner of the cart page.

Once all the items you want to purchase are added to your cart, you can begin the checkout process. At the bottom of the cart page, click the Proceed to Checkout button. If you are not currently logged in, you are directed to a login screen where you can log in with an existing account or create a new account. After that, you are taken back to the cart page. If you had items in your cart (while logged in) from a previous visit, the store automatically merges the items currently in your cart with the items that were previously in your cart. In the event that the store merges your cart, after logging in an alert will pop up to remind you to check the items in your cart prior to completing the checkout process. When logged in, clicking the Proceed to Checkout button takes you to the checkout page. If this is the first order you have placed on the

store, a form appears asking for your billing address. If you have previously placed an order, the previously used billing address appears. If you need to edit an address, click the Edit link to the right of the address. If you need to add a new address, click the Add Billing Address button. If you plan on paying via credit card, it's important to use a billing address that matches an address on file with the credit card company. To avoid a delay in processing the order, it's also important to use a billing location that is local to the geo-location of the IP address from which you are placing the order. Some products charge tax based on your billing address. If any taxes are applicable for your selected address, you can see them on the right side of the page. Once you are satisfied with the Billing Address, click Continue to go to the next step in the checkout process.

The next step is to enter your payment and billing details. At this point you have two options to make a payment: directly through the store with a credit card or through PayPal. If using PayPal you have the option of paying by account balance, bank transfer, credit card, or eCheck. The most popular way to pay is direct through the store by credit card. The store currently accepts Visa, MasterCard, American Express, and Discover credit cards and debit cards. To use any of these cards simply enter the name on the card, the card number (digits only), the security code CVC or CVV (four digits on the front for American Express, three digits on the back for all other cards), and the expiration month and year. Once you verify the information is correct, you need to agree to the sales terms and conditions by checking the box. If you haven't read the terms of use yet, do so at this time. There is important information regarding the refund policy and other aspects of the store. The last step is to click the Purchase button. This takes you to the order status screen; if everything has processed correctly, there is a link to the "My Downloads" area where you can download your purchase. If there is a problem with your order, a message appears explaining the issue and possible ways to resolve the issue.

Managing Your Account

Once you have established a user account on the site and made a purchase, additional items become available in the My Account section to help you access your downloads, manage your orders, obtain support, and other various functions of the store.

Accessing Your Downloads

Once you place an order, the next thing you typically want to do is to download the item you just purchased. This can be done on the My Account → My Downloads page. The downloads section was recently rebuilt from the ground up to give a faster and more efficient experience, especially for users who have purchased a large amount of items. The first thing you see at the top of the page is a few options for searching and filtering the items in your list. If you have more than 10 items in your downloads area, your results will have pagination and you have the option to specify how many results you want to show per page. The default is 10. The next option is the ability to filter the results by seller or vendor. This list is only populated with vendors from whom you have purchased an item in the past. The next item you can search by is package name. It's important to note that this returns partial matches, so if, for example, you are looking for a forms module that you purchased in the past but don't remember the exact name, you can try searching for simply the word "forms." The final option is the time frame in which you purchased the item. Options include all, last week, last 30 days, last 90 days, last 6 months, and last year. To apply the search or filter, just click the Filter/Refresh button. If you want to remove your filter and return to the default view, click the Reset link.

Each result displays the same basic information. On the left side there is product and order-specific information that includes the product icon and title, both of which link back to the Product Details page. The vendor name is listed for each item; that links you back to the vendor profile. Below that are the Invoice ID and the date the item was purchased. It's important to note that if you purchased multiple copies of a product via different orders, the product is listed multiple times on this page. That is because each time that a product occurs in your orders it is shown on this page so that you can view information for and interact with options in relation to each of the orders. In the right-hand column, you see a few links that can be quite helpful. The first link is the View Downloads link. When clicked, this opens a new page that lists each download available for that particular product. Each file displays the filename, file date (the date it was added to the system), and a download link. Each file can be downloaded by clicking the Download link in the far-right column. Back on the My Downloads page, the next link is the Get Help link. When clicked, this takes you to the store help desk system and automatically begins to create a help desk ticket for you for that particular product. This is the fastest way to get post-sales support from a vendor for that product. The help desk system is covered in more detail in a future section. The next link


on the My Downloads page is [View Receipt](#). When clicked, a receipt for the order appears. The receipt is covered more in the next section. The last item in the right-hand column of the My Downloads area is dynamic and related to your review of that item. If you haven't posted a review for the item, you see an [Add Review](#) link. Click it to open the [Add Reviews](#) page, which is covered in greater detail in a future section. If you have already posted a review for that product, there is an [Edit Review](#) link, and below that the number of stars you gave that product in your review on a scale of one to five. Click the [Edit Review](#) link if you want to edit any of your previous reviews.

Viewing Past Orders

Once you place a few orders, it's sometimes necessary to look back through previous orders. This is done on the [My Account](#) [My Orders](#) page, where you can search your past orders by using the order search at the top of the page. By default, all orders from the past year are shown. You can change the results by changing the date range in the search area. The results support sorting by clicking the column header by which you want to sort. By default, it's sorted by date descending (most recent order first). If you have more than ten orders, there are page numbers at the bottom that you can use to page through the results. Completed orders have a [Receipt](#) link to the right that when clicked displays a receipt for that order. This can be useful for accounting and record keeping. The receipt contains much of the same information found in the details area; however, the receipt displays it all on one page and makes it easier to print it. Click the [Details](#) link in the far-right column to open the order details section for the selected order. This area has three tabs—[Order Information](#), [Billing Information](#), and [Order Items](#). The [Order Information](#) tab includes invoice number, payment status, order date, and order total. There are also links on this tab to view the receipt or to email the receipt. The [Billing Information](#) tab includes the name, postal address, and email address for this order. The [Order Items](#) tab shows a list of all the items included in the order. Information shown here includes the name of the item, the name of the option if applicable, the quantity, price, status, and date downloaded.

Getting Support

There are a few avenues for support on the store. The primary page to help guide you to the correct method of support is the [Help Center](#). There is a link to this page in the primary navigation on the store. Often, the preferred

method to obtain support is the help desk. Based on your question or issue, there are a few different ways to submit a ticket. If you need to obtain presales support for a specific product, the best way is to click the Ask a Question link on that product's detail page. If you want to ask a specific vendor a question, the best way is to click the Ask A Question link on that vendor's profile page. If you need post-sales support for a product that you purchased, there are a few ways to submit a ticket. If you have the receipt email, to the right of each product is a Tech Help? link. You can click that to create a new help desk ticket for that product. Alternatively, you can click the Get Help link in the My Downloads area to accomplish the same thing. The final way to add a ticket is via the Help Center  Add Ticket page. This page has a drop-down menu where you can select from the following options: Product Name, Order Assistance, Login Assistance, Info Request, Downloads, Payment Inquiry, General Request, and Feedback. Selecting any of the items other than Product Name and clicking Go takes you to a page where you can submit a help desk ticket that is sent to the store support team for review. Selecting Product Name and clicking Go takes you to a page where you can either search for a product by name, or you can pick a product from a list of items that you have previously purchased. Once you have the product for which you want to obtain support, simply click the Get Help button. It's important to note that in order for the store to show a list of previously purchased items, you first need to log in.

Ultimately, all of these methods take you to the Add Ticket page. Depending on how you arrived at the page and whether you are logged in, a few of the fields may be prepopulated. If you are not logged in, you need to provide an email address and validate by filling out the captcha code. Another field that everyone will see is Short Description, which is a brief one-line description of the issue. This is similar to the subject of an email. Description is the main body of your question. This is a long-form text box, where you can ask your question in detail. Below that are two sets of radio buttons. If you want to attach a file (screen shots, code files, and so on) to the help desk ticket, select Yes next to Add Files. You will be prompted to upload your files once the ticket has been submitted. The last field is Allow FAQ. This is an opt-out field. If you do not want the vendor to show your question in the FAQ section of the Product Details page, you can opt out by selecting No. When you have filled out the form and are ready to submit it, click the Add Ticket button at the bottom of the page.

If you select the option to upload files, the Manage Files page opens. From here you can click the Browse button to select a file from your local computer and then click the Upload button to add it to the ticket. You can add multiple files by repeating that process. When you finish adding files to return to the ticket, simply click the Return To Ticket button.

Once you create the initial ticket and optionally upload files, you are redirected to the Edit Ticket screen. After a ticket is created, this is the main screen by which to interact with that ticket. Once the vendor or store support staff replies to the ticket, that reply is posted on this page. There are a few items on this page that are worth pointing out. At the top is a bit of a menu that allows you to perform the following actions:

- **Back to Main:** Click this link to return to your help desk ticket list.
- **Escalate Ticket:** Click this link if you feel that the vendor is not providing adequate support and want to bring it to the attention of the store staff.
- **Close Ticket:** Click this button if your question has been answered in a satisfactory manner.
- **View Files:** Click this link if you have upload files or want to upload a file. You can access the file management area. In the event that there are files associated with the ticket, the link also displays a number that represents how many files are attached.

Below those links is the Add Response area. From here you can type a text response and click the Add Response button to submit what you entered and add it to the ticket details. The Ticket Details area shows the correspondence related to this ticket. Each entry has a name and date/time stamp along with the response. The items are displayed in sequence with the most recent correspondence at the top.

The last section on this page is the Feedback area. If you want to rate the level of support given, you can select Positive, Neutral, or Negative, and then provide additional comments regarding the quality of the support in the Comment field. Click the Submit Rating button to send the feedback.

To see a list of and manage all the tickets you have submitted go to the Help Center ➤ Help Desk ➤ My Ticket List page. By default, any ticket updated or created in the last week is displayed. If you want to go back and search older tickets, you can select a different date range and then click Search. Basic

information about each ticket appears below the search, which includes the ticket number, short description, name of who created the ticket, the date it was created, the name of the person who last updated the ticket, and the date it was last updated. In the far-right column is an Edit link that takes you to the Edit Ticket screen, which was covered in the previous section.

Posting Reviews


Reviews are an important part of any shopping experience; it's helpful for other customers to be able to research the pros and cons of a product as evaluated by other users. It's also an important mechanism to voice your likes and dislikes about a product back to vendors so that they can improve their products to meet the market demands. The store tries to keep the review process as simple as possible. Recently, the process was updated to include an automated review reminder email that is sent to every customer three weeks after purchase. In addition to accessing the review system via that email, a quick way to post a review is from the My Downloads page, as discussed in a previous section. Finally, any customer can access the reviews system by logging in and then going to the My Account page. Under the Account Features column is an Add Reviews link. Clicking that link opens the main page where you can post reviews. Currently, the ability to post reviews is limited to customers who have purchased the product they are going to review, and a customer can review a product only once. Although it is possible to edit an existing review, we cover that in more depth later. The Add Reviews page shows a list of all the products for which you are able to post reviews. The product information includes the icon and product title in the first column. To the right, the next column contains the star rating scale. One to five stars can be assigned to the product (one being very poor and five being very good). The third column to the right is a freeform text box where you can leave a comment about the product. If you have multiple items in the list, you don't have to submit a review for every product. If you don't select a star rating, when you click the Submit Reviews button, no review is posted for that item. Before clicking the Submit Reviews button, read and agree to the statement at the bottom of the Reviews page.

To edit or remove a review, first log in and then go to the My Account page. Under the Account Features column is the Edit Reviews link. Clicking that link opens the Edit Reviews page where you can post reviews. This page displays a list of reviews that you have posted to the store. The left column contains the product name that links back to the Product Details page. Below

that is the vendor name, which links back to the vendor's profile. In the right column there are two links. Clicking the Edit Review link opens a modal pop-up where you can edit your review. Clicking the Delete Review link removes the review from the system.

Using the Watch List

For frequent customers of the store and for anyone looking to keep lists of commonly used products or even anyone who wants to keep an eye on a product for potential updates or price changes, the store now has an enhanced watch list function. If any of the items in the list have their price reduced or have a new version added, the system sends you a notification email. To access and use the watch list, you need to be logged into the store. As discussed earlier, products can be added to the watch list from the Product Details page. Each product page has a link in the bottom-right side of the product details area. Clicking the Add To Watch List link opens a modal pop-up where you can select the Watch Category (more on this to come) to which you want to add this product and also add an optional note. When ready, click Save to add the item and close the dialog, or click Cancel to just close the dialog.

To manage the items in your watch list, while logged in simply go to My Account  My Watch List. The Manage Watch List page first displays a drop-down list of watch categories (by default, everyone will have Uncategorized). If you want to add a watch category, simply click the Add Category button. The modal that opens asks for the name of the category. Although the categories can be used however you see fit, the intended use is to allow you to group watched products in the collections that could represent products to be used for a particular project or client. The watch list grid displays a list of products in a three-column format. The first column contains basic information about the product, including the title and icon, which both link back to the Product Details page. It also includes the name of the vendor, which links back to that vendor's profile page. Other information includes the date the product was added to the store, the base list price, and the product's short description. The second column contains any notes that you provided when adding the item to your watch list, and the last column includes two links (Edit Item and Remove Item) for managing the items. Remove Item will do just that. Clicking it removes the item from your watch list. Edit Item opens a modal pop-up that allows you to change the Watch Category to which

the item is assigned. You can also update the notes for the item.

The Referral Program

There is an opportunity with the store for people in the DNN ecosystem (even if you're not a module or skin developer) to earn a little extra income thanks to the store referral program. If you have a DNN-related site or personal blog site or you are a vendor who sells on the store but also maintains your own site to promote your products, you can earn a 10 percent commission on any sales that you refer to the store through the referral program. The best page to learn about the referral program is at [Make Money with DNN Referral Program](#). This page includes a link to the Referral Program FAQ and the Referral Program Agreement. If you are interested in participating in the referral program, it's important to read both pages, as they will help familiarize you with the details of the program.

Managing Your Referral Profile

If you want to participate in the referral program, the first step is to fill out a referral program profile. This is done by clicking the Referral Program Sign-Up link from the Referral Program page. If you're not currently logged in, this page prompts you to log in. The form asks for contact information. There are a few questions regarding the site that you intend on using to refer visitors to the store and finally a few questions regarding your preferred method of payment. Users who are just getting started with the referral program should consider using PayPal to receive payment, as it's typically the simplest to configure initially. After you complete your Referral Profile, click the Register button to submit the form. You should receive a confirmation email upon submission. Referral program registrations are typically processed within 24 hours unless it's a weekend or U.S. holiday. The more information you provide in the profile (specifically as it relates to the website where you intend to use the referral program) the quicker your application will be processed, and the more likely it is that your application will be approved. Once reviewed, you will receive an email notifying you regarding if your application has been approved or rejected. If approved, the acceptance email contains additional information on how to access your referral codes and how to begin using them. Once accepted into the referral program there are two more links at the bottom of the Referral Program page, which is covered next two sections.

Referring Sales

The referral program works by adding a special referral ID to each link on your site that directs users to the store. An example of this code would be F9EFED73E34A41838E93. Each code is unique and is a 20-digit hexadecimal number. Everyone approved into the referral program is given five different codes. You can use a different code on different sites or areas of your site, and via the reporting system you can see which codes are resulting in more referrals. There are a few ways to add a referral code to a URL. A few examples of how to add a referral code to a known product page are shown here. The following URL is for the DNN Corp. Annual Training Subscription:

<http://store.dnnsoftware.com/home/product-details/dnn-corp-annual-training-subscription>

Here are three different ways to link to that same page while also attaching the referral code:

<http://store.dnnsoftware.com/home/product-details/dnn-corp-annual-training-subscription?r=F9EFED73E34A41838E93><http://store.dnnsoftware.com/home/product-details/dnn-corp-annual-training-subscription/r/F9EFED73E34A41838E93><http://store.dnnsoftware.com/home/product-details.aspx?PackageID=22863&r=F9EFED73E34A41838E93>

As you can see, the referral code is just an additional query string parameter, so any store URL can have that parameter added when referring people to the site regardless of the destination page.

DNN Referral Module

Recently, a new free and open source module has been released that should simplify the process of integrating store product data with your referral code on your site. The DNN Referral Module is available in the forge and on CodePlex at <https://dnnreferralmodule.codeplex.com>. This module allows you to select which products to display. For example, it can be configured to display the top module or themes list. It can be configured to display all products from a vendor based on vendor ID. It also supports dynamic and static search methods. Dynamic allows it to listen to a specific search parameter and displays relevant results; static allows you to specify in the settings the word or phrase on which you want to base the search. The module also supports template-based output rendering so that you can configure the look and feel of the output to match the look and needs of your site. Once configured with your referral code, all the links are automatically generated with your referral code, and you'll never have to worry about an

invalid URL or outdated product data again.

Referral Reports

When your referral code is in place it should only be a matter of time based on your site traffic before you get your first referral. When an order is placed on the store from a user your site referred, an email is sent informing you of the sale. Once orders start to be placed using your referral codes, you can use the Referral Reports page to view info about those referrals. On this page you can view an order summary that provides basic information on completed sales made using your referral code. By default, only sales from the last two weeks are shown; however, the search date range can be expanded to include more results.

Below that form is a Sales Summary by Year report. Using that report, you can select a calendar year and view based on referral code how much sales revenue and commission were accumulated by month.

Selling on the Store

If you are a developer or designer who works with DNN professionally or perhaps only as a hobby, it's possible for you to become a vendor on the store. The store has vendors from around the globe. While creating and managing products for DNN is a full-time job for many vendors, there are even more who build extensions as a hobby and use the store to raise extra beer money. If you have an extension that you are interested in listing on the store, your first stop on the journey to becoming a vendor should be the [Make Money Become A Seller](#) page. This page is your best one-stop shop for all the information you need to get started.

Why become a vendor on the DNN Store? Listing your product in the store is the quickest way to tap into the more than 700,000 websites that have been deployed worldwide using the DNN Platform. By using the DNN Store as your sales channel, you can focus on your products while the store focuses on managing the e-commerce system, credit card and PayPal transaction processing, and fraud protection. The DNN Store also includes a help desk system for use when supporting customers of your product. The DNN Store actively markets products via weekly newsletters and Google AdWords. Listings on the DNN Store receive high organic search rankings due to the site's authority and SEO optimization.


With all of these features it's possible to become a vendor without even needing to maintain your own site for your business and products. If you choose to maintain a site independent of the store and even seek additional avenues to promote and market your products, then the DNN Store has a few features that can help. For starters, as covered previously, if you participate in the referral program, you receive an additional 10 percent commission from every sale you refer to the store. As a vendor, you can also configure advanced Google Analytics features to help track your sales funnel and the effectiveness of marketing campaigns. The store also supports Instant Order Notifications that can be issued from the store to your site upon the sale of a product for use in licensing or CRM workflows.

Becoming a Vendor

Once you make the decision to list a product on the store, the first step in becoming vendor is to fill out the vendor application. This can be done by clicking the [Become a Seller](#) link on the [Make Money Become A Seller](#) page.

If you are not currently logged in, the page will prompt you to first log in. When logged in, the page displays the seller application. It asks for basic contact information, and if you are in the United States, you need to complete the W-9 tax form-related fields. If you are not located in the U.S., you need to complete the W-8BEN fields. Most of the fields are going to be self-explanatory; however, it's important to point out that the Display Name is what will be used as your vendor name on your vendor profile page and on your products detail pages. When you complete the form, click the Update button at the bottom of the page. Upon submission of the vendor application, your account is automatically added to the vendor role, which gives you access to many new features on the My Account page. While your account is pending approval, you can make updates to your profile and begin to create product listings. In fact, doing so increases the speed and likelihood that your seller account will be approved.

Listing a Product

The next step in becoming an active vendor on the store is to create a product listing. This can be done via the My Account  My Products page. This page displays a list of every product you currently have listed on the store. If you haven't yet created any listings, then this page simply prompts you to create one. Click the Add A New Listing button to start this process.

On the Initial Create Product form you are asked for three basic bits of information about your product. First, what is the product name? This is the name used when your product is shown in the search results and product lists throughout the site. Also, the site ensures that this name is not already in use in the system by another product. The next field is the price. Technically this is the product “base price,” and it can be modified by product options that are covered in more depth soon. The amount entered must be a positive amount and expressed in USD. The third field is the short description. This field has a maximum length of 300 characters and is also shown in the search results and product lists throughout the site. This field is text only. No HTML or markup is permitted.

After you fill out the three fields, you need to read and check the box stating that you agree to the terms and conditions of the site. It's important to take the time to thoroughly read this agreement, as it outlines specific policies regarding product listing, seller responsibility, and refund policies.

Once submitted, you are redirected to the Edit Product Details page. This page is where you can manage everything related to your product. At the top of the page is a list of tabs: Product Details, Product Options, Product Downloads, Product Tags, Product Categories, and Product Specs.

The first tab, Product Details, contains the core information related to your product. It's important to fill this page out as completely and as accurately as possible. The first field on this form is a file upload for your product image. You should take the time to ensure that this is a good-quality image that accurately represents your product and your brand. This image is what is shown every time this product appears in the search results and other product lists throughout the site. It's recommended to use a 120x120-pixel JPEG file and no smaller than 85x85 pixels.

The next field, Product Name, should already be completed from the previous step; however, you can modify it if you desire. The next field is Product Type. This is used to classify your extension and is based on the supported extension types in the DNN platform. Currently you can select from Library, Module, Other, Provider, Skin, Skin Object, and Widget.

The next field is Min DNN Version. This is the minimum version of the DNN platform that your extension requires to function correctly. The next field is Max DNN Version, which is similar to the previous field, but the intention is that in the event there is a breaking change in the DNN Platform that will cause your extensions to no longer function correctly, the maximum compatible version can be set here. If there are no known breaking changes, then you can set this field to Current.

The next field is Unique Name. This needs to be whatever the unique name is that is specified in your extensions manifest file. It is important that this field matches up exactly as this data is used in conjunction with the update server and the product version number to notify DNN users in the extension management screen when an update is available for your extensions.

The next field is Product Version. As in the previous field, this one also needs to match what is specified in your extension manifest file. It also needs to be in the format of ##.##.##. So, for example, 07.00.01 would be the correct way to enter a version. The next field, Exclude From Extensions Feed, is a check box that allows you to opt your product out of the extensions feed. Starting with DNN 6.x the platform shipped with a new extensions gallery built in to the host functionality. This extension gallery pulls a product feed from both

the store and the forge. All listings are included in this by default, but if you want to exclude your product, you can do so by checking this box.

The next field, SKU or UPC, is an optional field where you can include your product's SKU or UPC code if you have one. The next field, Price, should also be predefined from the previous step; however, if you want to change it, you can do so at this time. The next field, Parent/Previous Version, is a field that is used to link multiple versions of a product that use a different product listing for each version. If this is your first product or this product is not a new version of an existing product, then leave this field set to none. However, if this product is a new version of a product that you currently have listed on the store, select that product from the list.

Linking a product to a previous version will do a few things. First, it causes a message to appear on the previous version's detail page that a new version is available and directs users to that new version. Second, it ensures that only the latest version is shown in the vendor profile and in the search results pages. Third, it links the sales from the previous version for purposes of ranking products based on sales.

The next field, Short Description, is another field that should already be prepopulated from the previous step; however, if you want to update it, you can do so at this time. The next field, Description, is one of the most important fields on this form and will likely be the one that takes the most time to complete. This field allows text and HTML markup and is the main description that is listed on the product details page. The content of this area is entirely up to you to provide. While composing the content for this field, you should look at the product pages of other vendors and figure out what information you feel is important to convey to the shopper.

Some suggestions for things to include are a product overview, list of features, demo links, screen shots, compatibility information, project history, comparisons between product options, and licensing. Ensure that the description represents your product in a good light, and try to customize the look and feel to match your company's brand. Please do not include references to external resources such as CSS or JavaScript files. If you are going to include images, it's recommended that you upload the images to the vendor file manager and then use those image URLs in your product description. Avoid JavaScript and CSS that will break the overall layout of the page or any of the functionality on the rest of the Product Details page.

The next field, License Overview, is an optional field that if specified displays your abbreviated license information at the bottom of the Product Details page. The last field, License, is for the full copy of your product license, terms, and conditions, if specified. This is shown at the bottom of your Product Details page. After you make changes to this form, it's important to click the Save button at the bottom before moving on to another section.

The next tab, Product Options, allows you to set up and manage different options for your product. An example of when you would use this would be if you had a module and you wanted to offer different license levels such as standard, professional, enterprise, and source. Another example is when you have a skin and want to offer the option to pick a color theme such as Blue, Green, Red, and so on. These are cases where you want to create one product with multiple options rather than create multiple products. Doing this reduces the number of listings that you need to manage while also driving more traffic and sales to one page, which will help boost your product's rank in search results and the top module or themes lists. Each option can have its own downloads and/or it can share downloads from the base package.

Options can also modify the base product price specified in the Product Details tab. This allows you to specify a different price based on the selected option. When you are ready to create an option, click the Add Product Option button. This opens a form inside a modal pop-up. The first field is for the Option Name. An example of this would be Standard Edition or Enterprise Edition. The next field is Price, which is a modifier to your product base price. It's important to note that a positive number increases the price of the product when this option is selected, while a negative number reduces the total price of the product (to a minimum of 0.00). This is useful if you want to list a trial or limited version of your product. The price is always expressed in USD.

The next field, Display (HTML), is what is shown in the Product Details page. This allows you to include images or the ability to style text with CSS. If you look at the bottom of the form, you see some examples of traditional options that display HTML values that are used on the store. The next field, View Order, is an integer that is used to rank the order in which the options are shown. It's recommended when assigning values that you leave gaps so that you can later insert items in the order if needed. For example, you could start by using values to the power of 10 (10, 20, 30, and so on). The form then asks if this is the Default option. When specifying options, it's important to set one

of them to be the Default option. This is the option that is automatically selected when a person arrives on the product details page. Also, if the Default option modifies the price of the product, then the modified price (base + Default option) is what will be used in the product lists, search results, newsletter, extensions gallery, and so on.

The next setting you can configure on this form is whether you want this option to be active. Typically, you will want the option to be active, but if you need to temporarily disable an option, setting this to No would accomplish that. The last thing that this form asks for is to specify how you want the store to handle downloads as they relate to the product and this option. You can choose to show users this option's download(s) only, show the user download(s) from the main product only, or show the user both the download(s) associated to the main product and this option. When you finish filling out the form, click Save. The page reloads and you should be able to then see the option listed in the table on the Product Options tab. You can edit or add additional options as needed using the same process as just described.

The next tab, Product Downloads, is where you can upload the files that you want customers to receive when they purchase an extension. If you have product options specified they appear in the options download. If you want to upload a file to associate with a specific option, first select that option from the drop-down list. If you want to upload a file to associate with the base product, then leave All Options selected and click the Choose File button. This opens a file picker dialog that you can use to browse your local computer for the file you want to upload. While you should always upload the installable package for the product you are listing, you can also upload other optional files such as a PDF or Word doc product manual. Or, if you are listing a skin, perhaps you want to upload the PSD file. Those can all be added from this page. It's important to note that once you select a file, you need to click the Upload button. This will begin the transfer from your computer to the server.

Once on the server the file is in a pending state until the server has had time to process it. This could take a few minutes depending on the size of the file, as a copy needs to be made and transmitted to Amazon S3 storage from where all files are pulled when a customer downloads them. While the server processes the file, you will see that the Processing check box is checked. When the file finishes processing, if you reload the page, you should see that box is no longer checked and the Has S3 box should now be checked. If the

file you uploaded is an installable package, you should check the Deployable check box. This allows people who have purchased this extension to auto-deploy it from within the DNN App Gallery. To download a file from there, click the filename. If you want to remove a download, click the red trash can icon to the right of the file.

The next tab, Product Tags, is how you can associate tags with your product. On the left side are two lists with popular and trending tags; you can associate any of these tags to your product by simply clicking them. Clicking them adds them to the box on the right. If you don't see any tags that you feel properly describe your product, you can add custom tags by typing them in to the box on the right. Press Enter to turn what you typed into a tag. Typing also opens an auto-complete where you can pick from existing tags in the system. If you want to remove a tag, click the red "x" to the right of the tag. When done, be sure to click the Save button to save your changes.

The next tab, Categories, is similar to tags, but these are predefined categories in the system where the tags are user-generated content. To add your product to a category, click the Add Product Category button. Clicking it opens a modal dialog that allows you to find categories that pertain to your listing. You can add them by clicking the Add link to the right of the category. It's important to note that you should always add categories to your products, but you should never put your product in categories that are unrelated to what your product does. You can only add ten categories to your listing, so if you find more than ten that apply, you need to prioritize the ones that are most important for your product. The limit is in place to prevent spamming products into categories that are unrelated.

The last tab is Product Specs. These are not a heavily utilized aspect of the product listing details, but they allow you to specify name/value pairs that describe your product. For example, you could do SQL Server Compatibility: 2005, 2008, 2012. These items will appear in the bottom-right side of the Product Details page below the spotlight reviews. To add a spec, click the Add Product Spec button. In the modal pop-up, the form asks for the Specification Name and the Description. In the previous example, the Specification Name would be SQL Server Compatibility, and the Description would be 2005, 2008, 2012. Once done, click Save to save and close the product spec.

When you finish editing your product details, you can see what your product details page looks like by going to the My Account > My Products page. Click the product title to see the details page. It's important to note that if the

product hasn't been published yet; only you and the store administrators can see the product page. Also, until the product is published you cannot add it to your cart. Back on the My Products page, there are a few features that are worth pointing out. Initially it won't be too useful, but as you add more products, the ability to sort and filter your product list will help you quickly find the product that you are looking for while making edits. By default, the results are sorted by Published Date, and ten items are shown per page. If you have more than ten items, you can page through them using the pagination at the bottom of the list. Or you can also change the number of items listed per page. Each item in the list displays the product icon, product title, your vendor name, short description, base price, and the date of the last update. This is similar to how your product will be shown in the search results and product list pages throughout the site. This page, however, includes a few extra management links for each product. Edit Listing takes you back to the Edit Product Details page previously discussed. The Copy Listing creates a new listing and is initially populated with data from this listing. A wizard asks you which information you want to copy over during this process. You need to give the product a unique name, and then you can specify if you would like to copy the options, tags, categories, and specs. By default, they are all selected. You also have the option to move the FAQs and Reviews to the new product. It's important to note that this occurs immediately, and if you choose to not move them at this time, it can be done at a later time. These options are not selected by default. Both FAQs and Reviews are covered in more depth in a later section. To finish the copy, read and agree to the terms, and then click the Save and Continue button. This will redirect you to the Edit Product Details page for the new listing. Again, back on the My Products page, there are two more links. Publish Listing validates your listing then activates it on the store. When you click this link, the store checks that your listing is complete and has a download associated to it. The system also checks to ensure that your vendor account is active. If your account hasn't been approved yet, you won't be able to activate your products. Once activated, your product will begin to show up in the store and will be available for customers to purchase.

The final management link, Delete Listing, de-activates the listing and removes it from your default view. Listings can be undeleted by switching the filter to Show deleted items. Additional products can be created by clicking the Add New Listing button in the upper-left area of the product list.

Managing Your Account

Now that you have the hang of creating and managing your products, you need to learn about the other auxiliary system included in the store to help vendors manage sales, discount customers, and support issues.

Managing Your Seller Profile

It's likely that you will want to visit your vendor profile and fill out some additional information soon after applying to be a vendor on the site. Only a portion of the vendor profile is configured on the initial sign-up form. You can access your seller profile by first logging into the site and then going to the My Account page. Under the Vendors – Sales column is the My Seller Profile link. Click it to open your seller profile. Near the top of the page is a list of tabs: Profile, Payments, Customer Message, Order Notification, Help Desk Settings, and Tax Information.

The Profile tab contains your basic fields for your seller profile. The first field, Supplier Name, should already be filled out with the information provided in the application. There are a few things that are important to note about this field. First, it must be unique; you cannot use the same seller name as another existing seller in the system. Second, this is the name that appears in the search results, product list, vendor profile, and product details page, so ensure that name represents your company or brand. Third, once your account is approved, you cannot change this name. If you want to change this after the account has been approved, you need to contact the store staff via the help desk system and request a name change.

The next field, Email Address, will be the email that is used as the primary point of contact for messaging from the store. Product sale, review, help desk emails, and so on, all go to this address. The next field, Alternate Email, allows you to specify an additional email address that receives only order notifications and help desk ticket notification. Some vendors find this useful because they can enter an email address for someone who might assist with sales and support issues. The next field, Slogan or Phrase, is an optional field that allows you to specify a company slogan that you want to appear on your vendor profile page. The next three fields are Google Analytics Account ID, GA SetDomainName, and Enable GA Ecommerce. They are covered later in the Analytics section.

The next field is the [Salesforce.com](https://salesforce.com) OID. This optional field is for vendors

who use [Salesforce.com](https://www.salesforce.com). By supplying your OID, the store transmits customer data upon order completion. The next section gives an explanation of how the Gravatar system is used on the store to display the vendor's avatar throughout the site. It's recommended that you set up a gravatar and associate it with the email address supplied previously. The final field, Product Header, is a text field that also supports HTML markup. Whatever you place in this field appears above your product description on every product details page.

The next tab is the Payments tab. It's important to take the time to complete this section and ensure that everything is correct, as the store is not able to pay you for product sales until this has been filled out. In the upper area on this page is a drop-down list where you can choose your preferred payment type. If you are in the United States, you can choose from Direct Deposit, Check, PayPal, and Wire Transfer. If you are not located in the United States, only PayPal and Wire Transfer payments are accepted. Another item worth noting is that a minimum account balance of \$1,000 USD is required before a payment will be made by wire transfer. Given that, the recommended payment method for new vendors would be PayPal as it is one of the quickest ways to get your account up and running. After selecting a payment type, click Update and then go to the corresponding tab in the lower section, complete the fields on that tab, and click the Update button. If you have a question about any of the fields on the Direct Deposit, Check, or Wire tabs, be sure to ask your bank how to obtain the correct information.

The Customer Message tab is an email message that is sent to everyone who purchases one of your products. The text box is the body copy of the message and supports the use of HTML markup. Below the text box is a check box. If checked, the system will BCC (blind carbon copy) the email message to you. Farther down the page is a list of tokens that can be used in the body of the email, and upon sending, the system replaces with information from the order. It's important to enhance this message to match your company's message and branding. It will leave a better impression with your customers than a simple text-only email. Once updates are made, click the Update button at the bottom of the page.

The Order Notification tab is used to configure the Instant Order Notification (ION) system. This system is used by many vendors to integrate with custom licensing solutions and CRM platforms. When an order is completed on the store, an ION POST is sent to the URL(s) specified in these settings. When

setting up the ION URL(s), be sure that the URLs are valid and begin with `http://` or `https://`. Test sends can be issued for each URL by clicking the Send Test button to the right of the URL. There is a third field for POST Security Value, which is an additional value that the store passes with the other fields in the POST so that you can verify that the POST originated from the store. ION POSTs are sent with data in the following form fields: InvoiceID, BillToFirstName, BillToLastName, BillToEmail, BillToState, BillToZip, BillToCountry, PackageID, OrderDetailID, PackageName, Quantity, Price, SKU_UPC, Discount, Amount, OptionID, OptionName, OptionText, and, if specified, SecurityValue. If you want to see a sample ION POST handler, there is a link at the bottom of this tab where you can download the demo. If any changes are made to the values on this page, be sure to click the Update button at the bottom of the tab.

The Help Desk Settings tab allows you to configure a few options regarding the help desk system on the store. First, there is an Out of the Office E-mail setting. This is the body copy of an optional out-of-office email that can be sent when a help desk ticket is created or updated. If you are going to be unavailable for an extended period of time, it's useful to configure this message so that customers understand why their ticket hasn't received a response and also to let them know when you will be available again to provide support. This field also supports HTML markup. The next field, Initial Ticket Reply Message, is an automated reply that can be posted to a newly created or assigned help desk ticket. This is a text-only message, and many vendors find this useful to specify your support hours and to set expectations of how long it will take to respond to the help desk ticket. Because the store has many vendors and customers from around the world, the times and days that you're available to provide support might not match up with the customers' expectations. These messages can help alleviate that issue.

The final tab, Tax Information, contains all the information the store needs to process any tax forms. Typically, unless you make over \$600 USD in referral commission and are based in the United States, your store income is not reported. However, you are still responsible to file and document any income from the store as required by any local laws that might apply to you. If you have questions about tax liabilities resulting from sales via the store, it is a good idea to consult a local tax accountant who would be knowledgeable in your area. The values in the Edit Tax Information form should already be filled out from the vendor application process. If you ever need to update your

tax profile information, it can be done from this tab. If changes are made, be sure to save them by clicking the Update button at the bottom of the page.

Viewing Your Sales

Once you start to make sales, you will find that occasionally you need to look up previous orders. This could be due to a licensing or support-related issue, the need to issue a refund, or simply curiosity. You can access your sales history by first logging on then going to the My Account page. Under the Vendors – Sales column, click the My Sales link to see the sales report. From here there are a few ways that you can look up sales. Click the My Current Sales button to show the sales and referrals for your account that haven't been paid out yet. If you need to look up an order from a specific person, you can look it up by using the Customer Name or E-mail search box. If you know the invoice ID, you can look up an order using the Invoice ID search box. You can also filter the results by transaction type if, for example, you know that you are looking for a product sale rather than a referral.

The results can also be filtered by date range using the Start Date and End Date boxes. After you enter or select your search criteria, click the Search button to get the results. If you want the system to generate an XML document from the results, you can check the Create Downloads check box. Once the search results are returned, a link appears at the top that can be used to download the XML file.

In looking at the search results, at the top you can see the system automatically adds up the sales total. The grid below that lists the transactions. Each row is a separate transaction in the system; information shown on this screen includes Invoice ID, Date, Product, Option, Type, Price, Net Amount, Customer Name, and, finally, a Details button. If you click the Details button for one of the transactions, it opens three new sections: Overview, Order Details, and Customer.

In the Overview section, you first find a Back to Search Results button, which when clicked takes you to the previous screen. Below that is the Sale Price for the item, the amount the customer paid. Below that is the Fees line, which is the portion of the transaction fee that goes to the store. Below that is the Net Amount, which is the amount that will be paid to you. Below that is the order status. Typically this will be completed, but it could also be pending or refunded. Below that is the Transaction Date, which is the date and time that the order was placed by the customer. The final item in this section is the

Invoice ID; this is the unique ID that is used throughout the system to reference this transaction. In the Order Details section is information about the product and option level of the item purchased by the customer. These fields include Package Name, Option, Option Text, Package ID, and Option ID. In the last section is basic contact information for the customer including Name, Email, City, State, Zip, and Country.

At the very bottom, if the order is completed and has been downloaded by the customer and the order date is within the 30-day refund window, you find the Authorize Refund button. If you want to issue a refund for this item to the customer, simply click this button.

Vendor Charts and Reports

There are a other charts and reports available to vendors. From the My Account page, under the Vendor – Sales column is a link for Statements. Click it to open the Statement Manager. Once you receive a payment from the store, you can run reports here by selecting a statement from the drop-down list. Optionally, you can filter the results by Customer, Memo, and Transaction Type. Clicking Search will show you any transactions that match your criteria. If you have a question about why a payment was for a particular amount, this is a good report to answer that question.

Back on the My Account page, there is a Reports link under the Vendor – Sales column. Click it to open the Vendor Reports screen. From here, you can select from seven different reports:

- **Product Sale Count:** Shows based on the date range how many of each of your products has been sold on the store.
- **Customer Details:** Shows based on the date range and product filters a list of customers. Data includes the order invoice ID, first name, last name, email address, country, product, amount, and date.
- **Customer List:** Shows based on the date range and product filters a list of customers. Data includes the customer's first name, last name, email address, and country.
- **Promo Summary:** Shows based on the date range and product filters a list of the promo codes used on orders during that period. The data includes the promo code, usage count, promo percentage, and product name.

- **Promo Detail:** Shows based on the date range and the product filters a list of orders that used a promo code. The data includes the invoice ID, promo code, promo percentage, first name, last name, email address of the customer, product name, amount, and date.
- **Statement Summary:** Shows based on date range the statements that include transactions.
- **Tax Collection:** If you have configured sales tax collection, this report shows how much was collected given the date range.

Back on the My Account page is a Vendor Chart link under the Vendor – Sales column. Click it to see the Vendor Charts screen, which displays charts that cover various metrics for the last 12 months.

- **Monthly Sales & Referrals:** Shows month-by-month your product sales, referral payout, and net revenue.
- **Monthly Sales Quantity:** Shows month-by-month totals for the number of items sold, the number of free items sold, and the number of paid items sold.
- **Average Transaction Amount:** Shows month-by-month what your average order amount is.
- **Discount System Utilization:** Shows a month-by-month breakdown of the total discount system utilization. There will be a total for coupons, discount package user, order package discounts, and sales discount created in the system.
- **Products:** Breaks down by month how many new products you have listed each.
- **FAQ System:** Shows the number of new FAQs you have created in the system each month.
- **Review System:** Shows the number of reviews added to the system for your products each month.
- **Help Desk System:** Shows the number of help desk tickets created each month.
- **Watch List System:** Shows each month how many times a product is added to someone's watch list.

Analytics

Most webmasters, online marketing professionals, and developers realize how important it is to be able to measure various metrics regarding website utilization. Page views, unique visits, visitor demographics, and tracking the sales funnel are things that can lead to a better understanding of your customer base and how they interact with your site and content. Ultimately, this understanding hopefully leads to making improvements to better address their needs and increase sales. Google Analytics has become an industry leader in this arena. The store utilizes GA tracking throughout the site. As a vendor, it is possible for you to also specify a GA tracking code in your vendor profile. If you specify a tracking code, the store renders that code on all your product details pages, your vendor profile page, and when customers add products to their carts and proceed through checkout, each step of that process utilizes the tracker code.

To enable basic tracking, you first need a Google Analytics account. If you don't already have one, you can sign up for one at <http://www.google.com/analytics>. Creating an account is quick, and best of all, it's free. Once you have a tracking code (example UA-98765432-1) log into the store and go to the My Account page. Under the Vendor – Sales column click the My Seller Profile link. About halfway down on the first tab is a field for Google Analytics Account ID. Copy and paste the tracking code that was assigned when you created your account. Click the Update button at the bottom of the page to save the change. Within 24 hours you should start to see usage data from the store in your Google Analytics account.

This functionality was upgraded about a year after the initial site relaunch to add support for GA Ecommerce Tracking. Basically, this feature transmits order information back to GA, so the system can then report on the value of a page or a search term. If you choose to advertise, this feature is useful to track the return on that advertising. To enable Ecommerce Tracking, log in to your GA account and then go to the dashboard for the code that you want to enable. Click Admin in the upper-right side of the screen. At the top of the third column select View Settings.

On this page is a setting to enable Ecommerce Tracking. Be sure that it's set to ON. Next, log in to the store and go to the My Account page. Under the Vendor – Sales column click the My Seller Profile link. About halfway down on the first tab is the Enable GA Ecommerce field. Check that box, and then click Update at the bottom of the page to save the changes. The next time that

anyone purchases one of your products on the store the ecommerce data will appear in your GA account.

Discounts

Discounts can be an effective way to drive product sales; they can also be a useful tool for rewarding customer loyalty. The store supports four discount systems: Product (Package) Discounts, Customer Quantity Discounts, Customer Discounts, and Coupons. This section covers how each of the four systems works, why you would use them, and how to configure them. To access the four discount systems and to see a high-level overview, log in to the store and then go to the My Account page. Under the Products column click the Discount Overviews link to open the discount overview page. You can access the four discount systems from this page.

The Product (Package) Discount system allows you to configure a discount to be given based on a customer's previous or concurrent purchase. There are a few different uses for this discount type. First, it can be used to provide an upgrade discount. For example, if a customer purchased the 1.0 version of your product last year and now you released the 2.0 version, you can configure a discount in this system to give anyone who is upgrading a percentage discount. Another use of this system is to provide discounts when moving from one option to another within a product. If, for example, you have a module that sells for \$100 and the source option sells for \$200, then you can set up a 50 percent discount for customers upgrading to the source version from the standard version. This way they would basically only be paying the difference. The third use of this system is to provide a bundle discount. An example of this would be if you purchase Product A, the user gets a percentage off of Product B. When you have products that complement one another, this is a good way to cross-sell products. To create one of these discounts, you need to go to the product discounts page by clicking the Product Discounts link on the Discount Overview page to display the management tool for these discounts. From there, click the Add New Discount button at the bottom of the screen to open a modal dialog with a form inside of it with a list of the fields and how to configure them:

- **Package:** From this drop-down list, you select the product for which you to create the discount.
- **Package Option:** If you want to restrict this discount to a specific option, select it from the drop-down list. If you want this discount to apply to all

options leave it as Select one.

- **Ordered Package:** This drop-down list contains a list of packages that can be used as the prerequisite for this discount. That means for customers to be able to use this discount this item needs to be in their order history or in their cart.
- **Ordered Options:** This drop-down list gives you the ability to restrict this discount to a particular prerequisite option. If you want this discount to apply to all options, leave the selected option as Select one.
- **Discount %:** This is the percentage that you want to discount from the retail price. This can be any whole number between 0 and 100.
- **Qualifying Start Date and Qualifying End Date:** These two fields work together to specify a date range for when the prerequisite item had to have been ordered for it to count for this discount. This is used to possibly restrict a discount to customers who have purchased the previous version within the last six months. Any purchases outside of that time frame would not count.
- **Is Active:** Discounts cannot be deleted; however, this check box allows you to deactivate a discount if you no longer want anyone to be able to utilize this discount.
- **Description:** This is a text-based memo field that can be used to describe the purpose of the discount. This field is optional and is for internal reference only.

The second system is the Customer Quantity Discounts. This system is an easy way to reward customers who buy your products frequently or purchase multiple quantities of your products. Based on a user's cumulative item purchase quantity, you can specify percentage-based discounts. For example, customers who buy ten or more of your products receive a 10 percent discount. These can also be multilayer, so you could set up another level for a customer who purchases 20 or more of your products gets a 20 percent discount. The item count is based on anything the customer has previously purchased from you as well as any items in the cart. These discounts are a good way to reward frequent customers. To create one of these discounts, go to the customer quantity discounts page by clicking the Customer Quantity Discounts link on the Discount Overview page. This takes you to the management tool for these discounts. From there, click the Add New

Discount button at the bottom of the screen to open a modal dialog with a form inside of it. Following is a list of the fields and how to configure them:

- **Order Count:** This field accepts a positive integer, which is the number of items a customer must order (or have ordered) to receive this discount.
- **Discount %:** This is the percentage that you want to discount from the retail price. This can be any whole number between 0 and 100.

The third system is Customer Discounts. This is a good way to give an existing customer a discount. With this system you can specify a percentage discount on a particular product for a specific user. It also includes the ability to restrict the discount to a particular product option. This system is best suited for providing an upgrade discount. Oftentimes vendors receive a help desk ticket from a customer requesting a discount on an item or an upgrade. Sometimes the customer may have purchased the incorrect option level and would like to get a discount to move to the correct one. If you choose to offer a discount as part of the post-sales support process, this system is the best way to accomplish that. To create one of these discounts, go to the customer discounts page by clicking the Customer Discounts link on the Discount Overview page. This takes you to the management tool for these discounts. From there, click the Add New Discount button at the bottom of the screen to open a modal dialog with a form inside of it. Here is a list of the fields and how to configure them:

- **Package:** From this drop-down list select the product for which you want to create the discount.
- **Package Option:** If you want to restrict this discount to a specific option, select it from the drop-down list. If you want this discount to apply to all options, leave it as Select one.
- **Select user by:** This allows you to pick if you want to select the buyer by Buyer Email or Sold Package. If Buyer Email is selected, a Buyer Email text box appears. Once you enter a valid email address, the E-mail field updates with a list of users that you can choose to associate this discount. If you select Sold Package, a drop-down list with your products appears. When you select a product, the option drop-down updates, and you can select a particular option to narrow your results or leave it as Select one. When a selection is made, the E-mail field below updates with a list of users who have purchased that product and option.

- **E-mail:** This is automatically populated based on the previous fields. Select the user from this list for whom you want to configure the discount.
- **Discount %:** This is the percentage that you want to discount from the retail price. This can be any whole number between 0 and 100.
- **Issued Count:** This is a way to limit how many times this discount can be used. If you set the issue count to 1, then this discount can be used only once.
- **Memo:** This is a text-based field that can be used to describe the purpose of the discount. This field is optional and is for internal reference only.
- **Send Email:** If selected, this tells the system to send an email to customers to inform them that this discount has been created.
- **Email message:** This is the body of the email that is sent to the user upon creation of the discount if the above Email check box has been selected.

The fourth system is the Coupons. This system allows you to create coupons that give users a percentage-based discount. These discounts can be applied at the vendor level (all of your products) or at the product level (one product, but would work for any of the options). These discount codes can be public or private. A public one is auto-applied at checkout by the store. A private discount requires the customer to enter the code at checkout. The coupons can also have a limited quantity and expiration date. These are useful to tie in with promotions and marketing campaigns. For example, you could have a holiday sale and offer a 10 percent discount on all your products. These sorts of discounts and marketing campaigns are a good way to push customers who might be on the fence about buying your product to finally making a purchase or buying an upgrade. To create one of these discounts, go to the coupon system page by clicking the Coupon system link on the Discount Overview page. This takes you to the management tool for these discounts. From there, click the Add New Coupon button at the bottom of the screen to open a modal dialog with a form inside of it. Here is a list of the fields and how to configure them:

- **Product:** From this drop-down list select the product for which you want to create the coupon. With coupons, you can also leave this set to <All> and the coupon will work with all of your active products.
- **Coupon Code:** This is the unique code that is used to represent your

discount. If the coupon is private, this is the code that you will need to provide to the customers in order for them to use this coupon.

- **Issued Quantity:** This is a way to limit how many times this discount can be used. If you set the issue count to 1, then this discount can be used only once.
- **End Date:** This is the date the coupon code expires. After this date, the code stops working.
- **Discount %:** This is the percentage that you want to discount from the retail price. This can be any whole number between 0 and 100.
- **Memo:** This is a text-based field that can be used to describe the purpose of the discount. This field is optional and is for internal reference only.
- **Public Promo?:** This check box allows you to set the coupon to public or private. A public coupon is automatically applied at the time of checkout to anyone ordering the product for which it is configured. A private coupon requires a customer to enter the code in the cart before checkout.
- **Promo Display Text:** This is the text that is shown to the user to describe the coupon.

Using the Help Desk

Using the help desk by the customers to obtain support was covered in a previous section. This section covers additional features available for vendors. For the most part, the help desk works the same for both customers and vendors; however, the help desk has a few extra features for vendors. These enhancements help simplify the process of providing support for your products. Previously, in the vendor profile management section, the settings that allow for out-of-office replies and automatic ticket responses were covered.

To see some of the other enhancements, go to Help Center, My Ticket List. There is a check box on this page to the right of the date filter boxes. When checked, Support Tickets means that the ticket list will show a ticket from customers who need your support. If you submit support tickets to the store or other vendors, deselect this box to see the tickets you've added to the system. Other than that change, the rest of this screen works the same as it does for the customers. If you have a ticket in your support queue, you can click the edit link in the right-hand column to open that ticket. On the edit

ticket page, one of the biggest differences you see is below the Response area, where there is now a Quick Text function. Clicking the bar expands the section and reveals the new functions. Even if you haven't used this feature before, there will already be a few default items that all vendors get. You cannot edit or remove these items, but you can add more.

To respond to a help desk ticket using a quick text, select the one you want to use from the list and click the Insert button. This injects the quick text into the response box. You can make any changes you need to the text, and when you are ready to post the reply, click the Add Response button. If you want to add a new quick text option, click the Add New button. This displays a mini form where you can enter the Quick Text Name, which is what is shown in the list box, and the Quick Text, which is the text that will be injected into the response box. After you fill out the form, click Save Changes to save your new quick text and close the form. You can then edit any item you create by first selecting it and then clicking the Edit button. An edit form opens and from there it works just like the Add New form. From this screen you can also delete any of the items you've created by selecting it from the list and clicking the Delete button.

Once you post a reply, if the customer has not opted out and the ticket is in context of a product, a Create FAQ from Ticket link appears at the top that allows you to turn this ticket into a FAQ. Clicking the link opens a new Create FAQ from Helpdesk Ticket form where you see two boxes. The first is the customer's original question, and the second is your initial reply. At this point you can make any needed edits to both fields. If you want the FAQ to appear on the product page right away, leave the FAQ Is Approved box checked. Deselecting this adds the FAQ to the system, but it will not be active. To activate it, you need to go to the Manage FAQ module, which is covered in the next section.

Managing FAQs

FAQs are a good way to list common presales questions on your product details page in an attempt to offer self-support for potential customers. To manage your FAQs, log in and then go to the My Account page. Click the Manage FAQs link under the Support column to display the Manage FAQs screen. From this page you can look up FAQs by date range, assigned product, and approval status. Clicking the Search button displays the results. From here you can do two things. First, if you want to move one (or multiple) FAQ

item(s) to a different package, check the Select box for each item you want to move; then, in the Move Selected Faqs drop-down list, select the product you want to move them to, and then click the Move link. To edit an FAQ, click the Edit button in the right column of the result grid. In the new screen that appears you can edit the Package, Question, Answer, and the Is Approved flag.

If you want to add a new FAQ, click the Add New button. A new screen appears where you can create a new FAQ. The first field is Package. Select from this list the package to which you want this FAQ associated. In the next field, Question, enter the text question portion of this FAQ. In the next field, Answer, enter the text answer portion of this FAQ. Finally, there is a check box for Is Approved. If you want this FAQ to be shown on the product page be sure to check the Is Approved box. Once the form filled out, click the Save button to save and close the form.

Managing Product Reviews

As you start to make sales, you will also begin to eventually receive product reviews. It's important to know how the review system works so that you can manage any reviews that your products might get. To access the review system log in and go to the My Account page. Click the Manage Product Reviews link under the Products heading to display the manage product reviews page. There is a list of all your product reviews in the system on this page. Use the Search bar to narrow the results by any of the fields below. Just begin typing and it starts to filter out results. If you want to move one or more reviews from one product to another, select the review(s) you want to move by checking the box in the left column. To move it, select the product from the drop-down list at the bottom of the grid and then click the Move Reviews link. To view the details of a review, click the Details button to the right of it. While you can't edit the review itself, you can add a reply, which is shown in the review area with the customer's review. The reply will also be emailed to the customer.

The other field that you can update is the Display. This determines whether the review is shown in the Spotlight Reviews area. It's important to note about reviews that customers can come back and update reviews. If a customer leaves a negative review, typically it's due to poor support or an unanswered question. If you contact the customer via the review reply and/or the help desk and help work out the issue, you can then ask the customer to come back and update the review. It's also important to note that only

customers who have ordered a product from you can leave a review on that product. There is also a delay of about two days from the time a review is posted to the time it first shows up on the site. This is by design to give you an opportunity to respond to the review.

The Vendor File Manager

While working on your product listing you might want to provide screen shots, product images, trial downloads, and so on. It's preferred that you use the built-in vendor file manager system to host those files rather than your own hosting or a third-party image hosting service. To access the vendor file manager, log in and from the My Account page click the File Manager link under the Products column. To upload files, click the Add files button to open a file picker dialog. Select the file(s) you want to upload and then click Open. Once the file picker dialog closes, you will see the items added to the file grid. You can then click the Start upload button to begin transferring the files to the server. As files are being uploaded, you will notice that some files are green while others are orange. Green files have been synced to Amazon S3, while orange files are still pending upload. If you have orange files, refresh the page until they turn green. Depending on the size of the file, it could take a little while. Once your files have turned green, you can obtain the URL by right-clicking the filename and selecting copy link location. Clicking the link opens (or downloads) the file. You can then use the links to the files in your product description or anywhere else on the store where you need to have hosted files.

Sales Tax Configuration

Depending on your location and your sales volume, you can configure the store to collect sales tax when a customer orders a product from an applicable location. To access the sales tax system, log in then go to the My Accounts page. Click the Sales Tax Config link under the Vendors – Sales column. The Sales Tax Management screen shows a table with all the taxes you have configured for your account. You can filter the results by using the Search box in the upper-right corner. The filter works automatically on all fields as you type. To add a new sales tax item to the system, click the Add Sales Tax button at the bottom of the results grid. The add tax form opens in a modal pop-up. In the modal, there are a few fields you need to complete. First, select the Country to which you want this tax to apply. Once the Country is selected, the Region list loads with possible regions within that country. If the tax is not

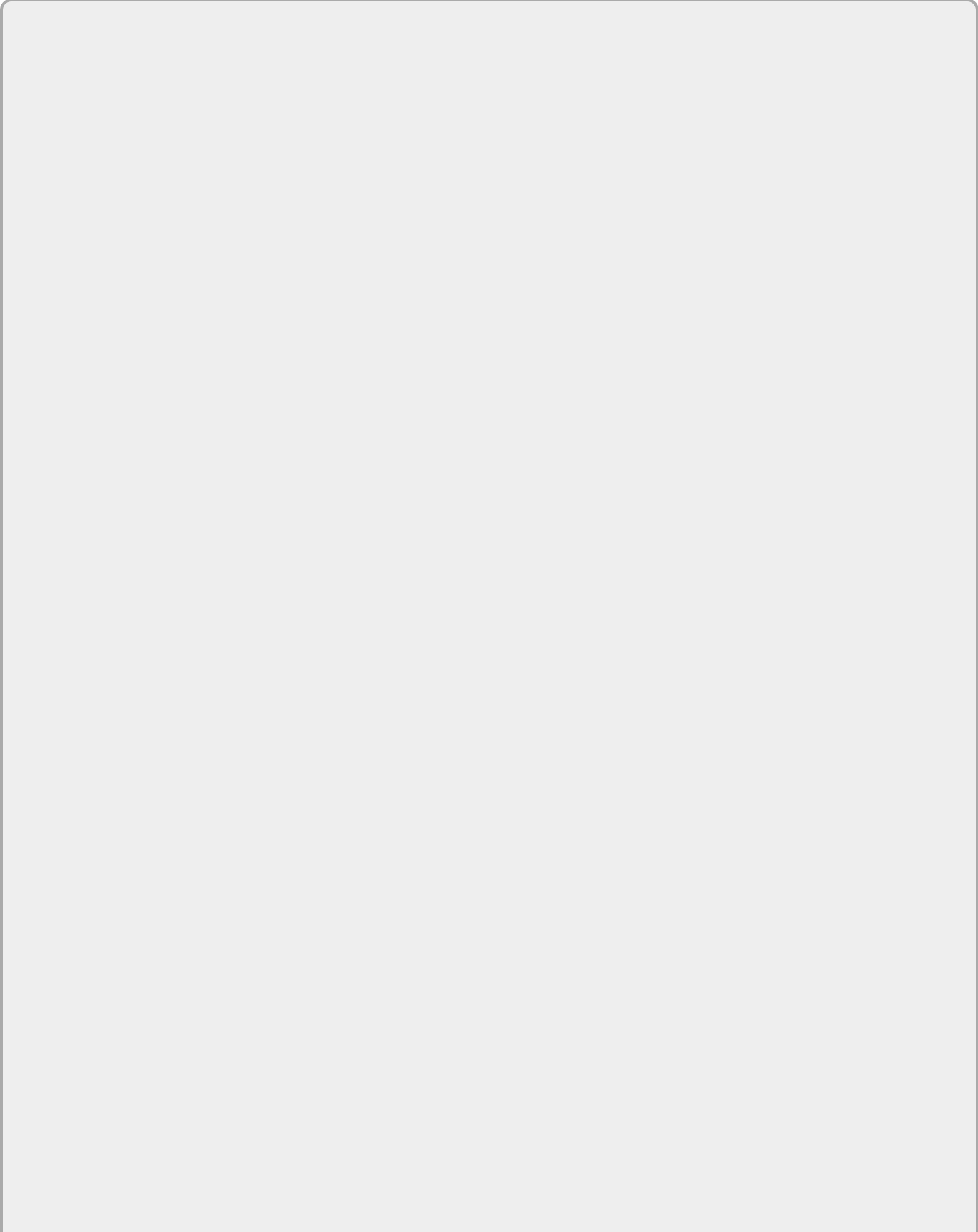
region specific, you can select the All option. If the tax is a local, postal (ZIP) code-specific tax, you can enter that in the Local Code box. Next, enter the Tax Rate percentage. This needs to be a decimal greater than 0.00. Next, you can specify a Use Order that will rank the order that taxes are applied if more than one is applicable. The next field is Tax Description, which is what will be shown to the customer to explain the tax. The final field, Notes, is an optional field that you can use to make notes about this tax item. This field is for internal use only. Once you fill out the form, click the Save button to save and close the modal. If you need to edit or delete a sales tax item, click the Edit link to the far right of the item in the grid.

Summary

The store is the official market place for the DNN platform. Due to the extensive nature of DNN, the store plays a crucial role in helping developers and users of DNN to quickly and affordably extend DNN to meet their specific business requirements. For hobbyist and DNN professionals, the store also offers a unique opportunity to earn extra income by participating in the referral program. For developers, the store can serve as a means to quickly and easily bring your DNN extensions to market by becoming a vendor on the store.

Chapter 23

DNN on Microsoft Azure



What You Will Learn in This Chapter

- Opening a Microsoft Azure subscription
- Deploying and configuring installations
- Creating backups
- Upgrading DNN versions
- Maintaining and managing your site

Wrox.com Code Downloads for this Chapter

The wrox.com code downloads for this chapter are found at www.wrox.com/go/prodnn7 on the Download Code tab. The code is in the [Chapter 23](#) download and individually named according to the names throughout the chapter.

When Microsoft Azure first appeared, many in the DNN community moved to work out how to deploy a DNN installation on the new platform. There were many issues to overcome around compatibility and suitability for the new cloud environment offering from Microsoft. Over time these challenges were all met, and Azure has evolved alongside DNN. Today it's simple to deploy DNN in an Azure environment and take advantage of the scalability, reliability, and flexibility of cloud computing. DNN is tested on the Azure platform every day and is compatible out of the box. The remainder of this chapter provides key information on getting the most out of DNN when installed on Azure.

Azure Deployment Scenarios

Three choices exist for deploying DNN on Azure. The choices are

- Virtual machines (VMs)
- Cloud Services
- Azure Websites

Hosting on virtual machines is very similar to hosting on a virtual private server or dedicated server. This model is called *Infrastructure as a Service* (IaaS). The virtual machine is persisted, and it's possible to install a SQL Server edition either on the machine or on a separate machine. Because it is a persistent virtual machine, the owner is responsible for applying software and system updates.

Cloud Services is the original *Platform as a Service* (PaaS) solution for Azure. In this configuration, a package is defined that allows the Cloud Services environment to create virtual machine instances with specified software configured on creation. A cloud service is always defined with a minimum of two instances to provide redundancy and increase uptime. Because it's a managed environment, the instances will be dropped and re-created periodically by the Azure environment, to apply operating system patches and updates. No persistent data can be stored on the local disks of the cloud service unless it is configured in the package. The owner is responsible only for defining the package configuration.

Azure Websites is the latest addition and is also a PaaS solution. Azure Websites is best described as "IIS as a service." You don't get access to a virtual machine with Azure Websites, but you can host multiple sites in a single website instance. As a PaaS solution, Azure takes care of the operation system updates, load balancing, and redundancy for uptime. Unlike Cloud Services, Azure Websites come with a set amount of persistent storage, and only the IIS process is dynamic.

The nature of DNN lends itself to Azure Websites deployments. DNN works well on the Cloud Services environment but takes more effort to configure to persistently store the application files. Persistent storage is required due to the ability for a DNN site administrator to modify the site and add new extensions after the site is first installed. This is possible with virtual machines, but the nature of the deployment is much the same as a regular

installation and does not warrant an entire chapter.

The remainder of this chapter describes how to deploy DNN on Azure Websites as the preferred option of the available deployment scenarios.

Installing DNN on Azure Websites

DNN is available on Azure Websites as a pre-installed application. This makes it simple to get started with DNN. It's always on the latest version, because the release process for new DNN versions includes providing Microsoft with the DNN installation files for inclusion in the Azure site.

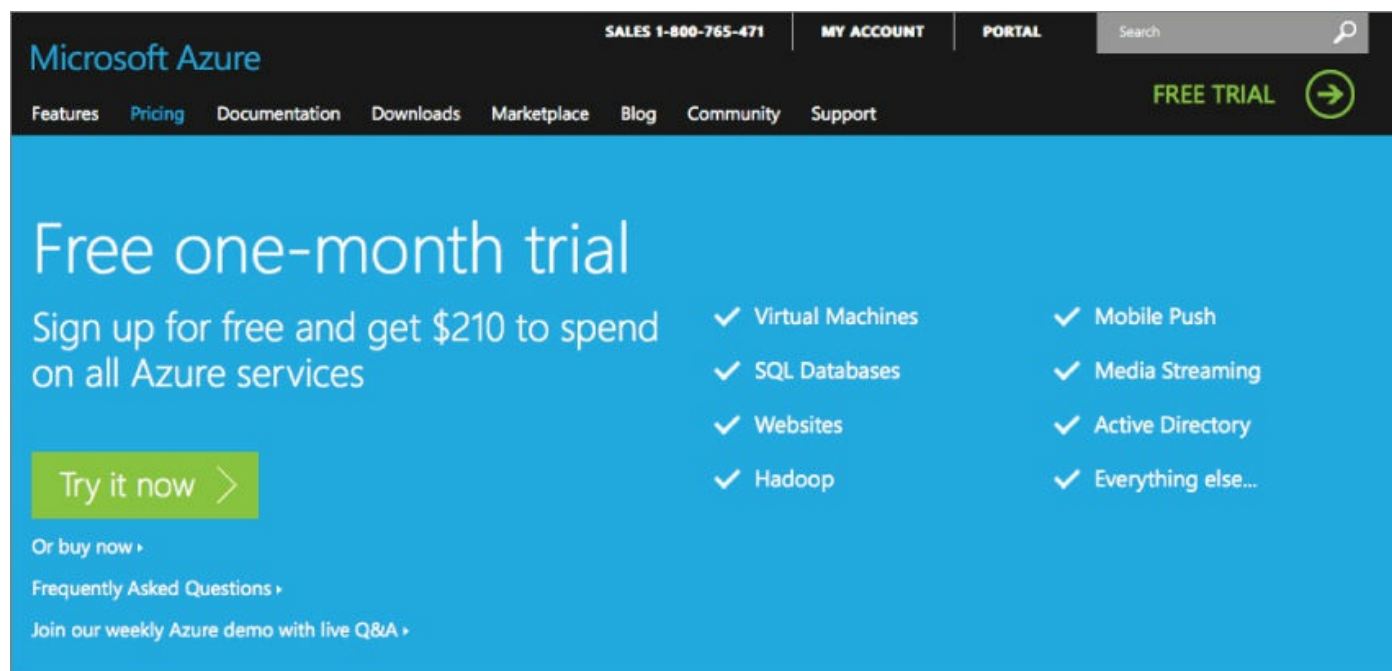
The following sections explain how to set up a new subscription and log into the new DNN installation for the first time.

Install Preparation

Before diving into the installation, you need to follow a few preparation steps to ensure you have access to Azure and to get everything ready.

Preliminary Step 1: Opening a New Azure Subscription

Microsoft Azure accounts consist of a Microsoft account (such as a Hotmail, Outlook, or Live account), which is associated with one or more Azure subscriptions. An Azure subscription is a container in which different resources can be added. The subscription determines the price and billing details for the resources. It's usually possible to open a new subscription with a free trial period, as shown in [Figure 23.1](#). You can download the free trial from <http://azure.microsoft.com/en-us/pricing/free-trial/>.



[Figure 23.1](#)

Within the signup process, choose your country/region carefully. The selected value will determine the billing currency and other subscription conditions, such as which Azure regions are available. This cannot be changed once the subscription is created, so choose carefully.

You will need to verify with an SMS message or automated phone call, using a device linked to your Microsoft account. Have this ready before starting the signup process.

Once you have a subscription to work with, you're ready to start creating Azure resources.

Preliminary Step 2: Starting a Credentials File

While you're creating the resources to support your new DNN site, you'll be creating a lot of credentials and resources. It's good practice to write these down in a text file as you go, so it's easy to refer back to a specific credential when you need it.

These are the values you'll collect, so create the headings in a file:

- Website name:
- Azure region:
- Azure SQL server name:
- Azure SQL database name:
- Azure SQL login username:
- Azure SQL login password:
- Azure website URL:
- DNN host username:
- DNN host password:
- FTP endpoint:
- FTP user:
- FTP password:

You can save the file in a secure location when you're finished, which will ensure that you have a copy of the key credentials for future use.

Performing the DNN Installation

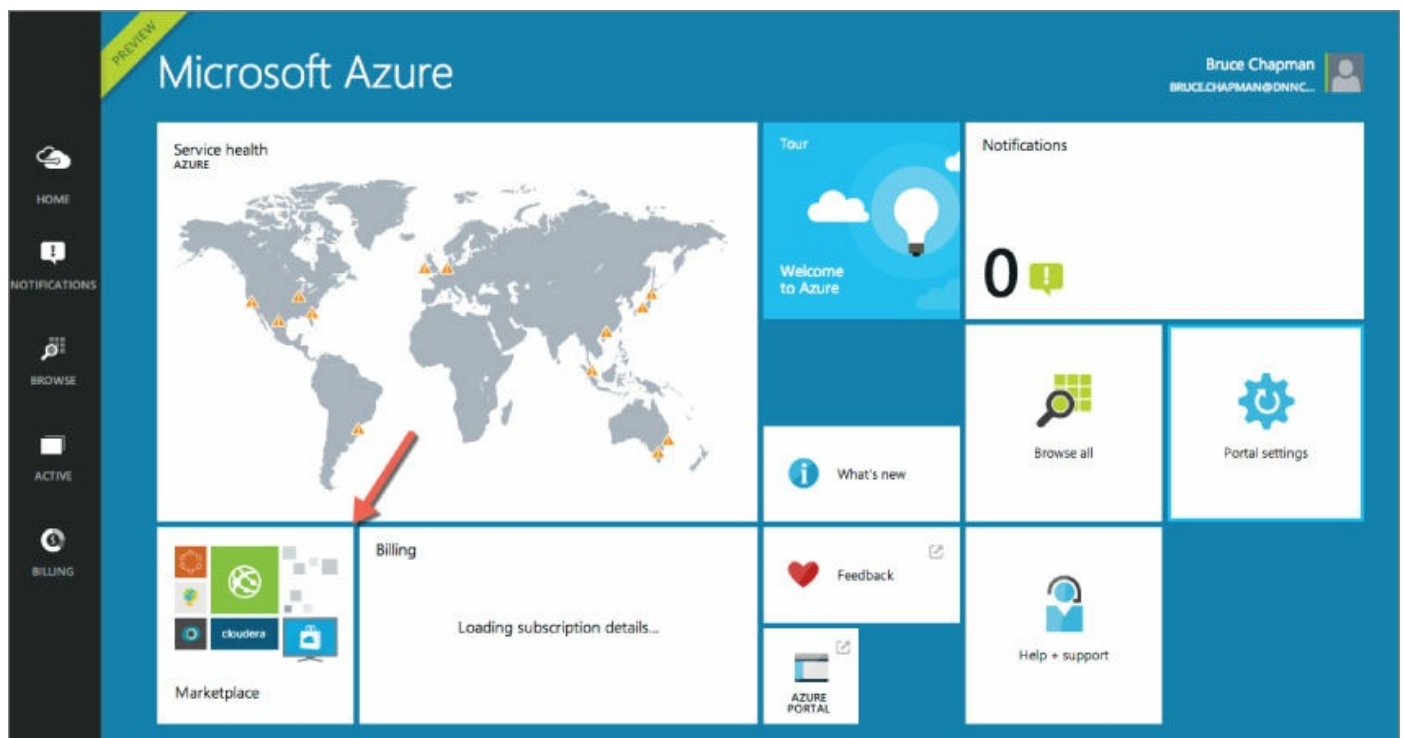
Once access to Azure has been organized and your credentials file is ready, it's time to start running the installation process. DNN is available from the Azure Marketplace as a free installation.

Step 1: Opening the Azure Site and Creating a New DNN Install

Navigate to the Azure site at <http://portal.azure.com/>. At the time of writing, the Azure site was still in preview, but Microsoft has indicated that all new features will be released on this site. The remainder of the chapter shows the new site. The Azure site has a design language using *blades*, which are context-specific windows of data that relate to the previously selected items. You open them by horizontally scrolling across the page, and tiles in one blade tend to open another.

Wait for the page to complete loading, and then start by clicking on the marketplace. This loads the Marketplace blade. From there, select Web to show the Azure Websites options. Each product available in the marketplace shows a tile—it's easy to pick out the DNN platform tile and create from there.

[Figure 23.2](#) shows the Marketplace tile.



[Figure 23.2](#)

This opens another blade showing the description of DNN—at the bottom

you'll see a Create button. Click this button to start the process.

[Figure 23.3](#) shows the DNN platform choice in the Marketplace blade and the location of the Create button.

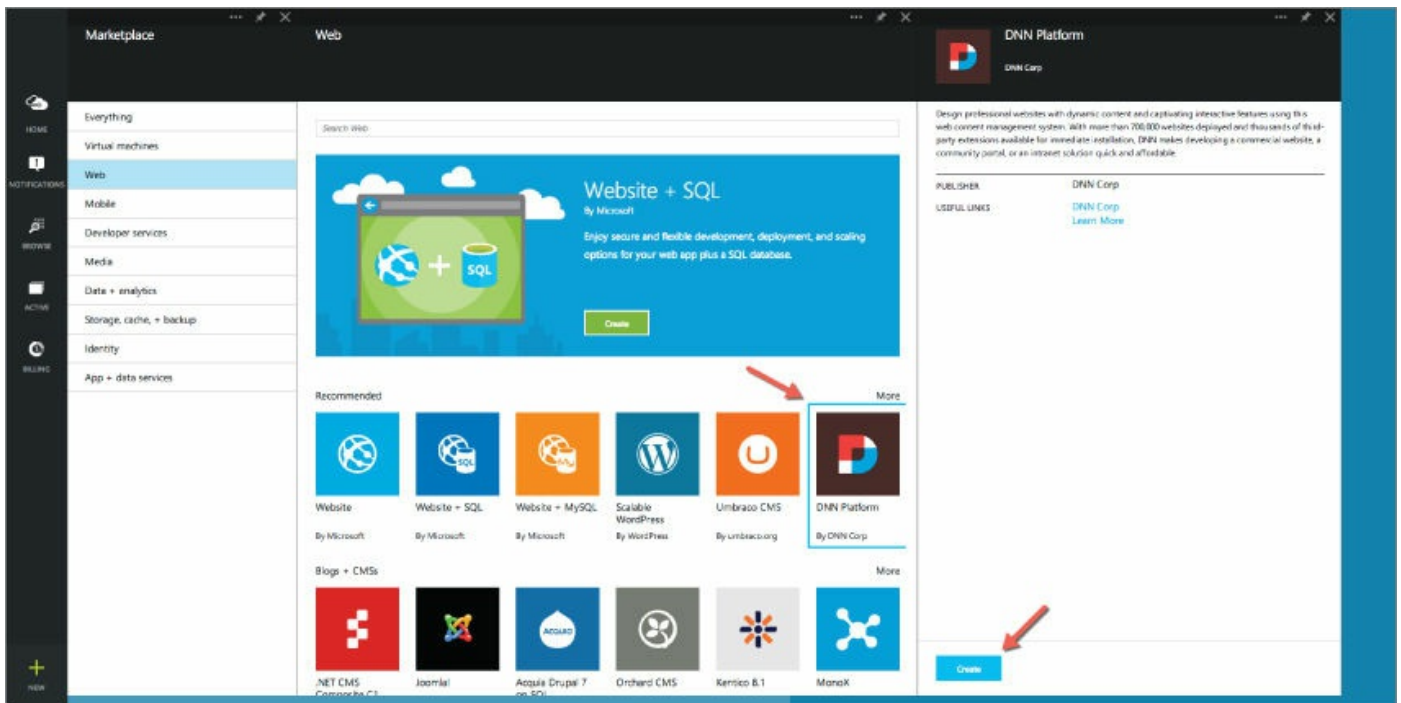


Figure 23.3

Step 2: Choosing the Resource Plan for Your Azure Website

Before you can complete the creation of your Azure website, you must choose the various levels of resources and set configuration options.

First choose a name for your Azure website. This name will appear everywhere—including in the default URL—so choose something descriptive. This example uses `dnn7demo`.

Websites live in a resource group, so this is the first thing to be named. If you're going to install a single Azure website, we recommend giving the resource group the same name as the website (in this case, `dnn7`). If you're going to create multiple sites, you might consider a different name (such as `Sales Dept Sites`).

Within the resource group, you will see that Azure has selected Website (the Azure website), Database (the SQL Azure database your site will use), and Subscription (the subscription it will be created in, which determines the billing).

Step 3: Creating the Azure Website

After you've named the resource group, click the Website selection. You can enter the URL value, which will translate into a finished site default URL of `http://{url}.azurewebsites.net/`. Enter the value here—this example uses `dnn7demo`. As soon as you get confirmation that the name is available, you can copy the name to the Website Name: header in your credentials text file.

This provides yet another blade to be configured. This is the Web Hosting Plan blade, which is a container for your Azure Websites resources. Again, you need to enter a name. You can depart from the standard naming approach and use a `-plan` suffix to make it distinct. [Figure 23.4](#) shows the creation process with the resource group, name (URL), and selected pricing tier of the Azure website. In this example, the name `dnn7demo` is being used.

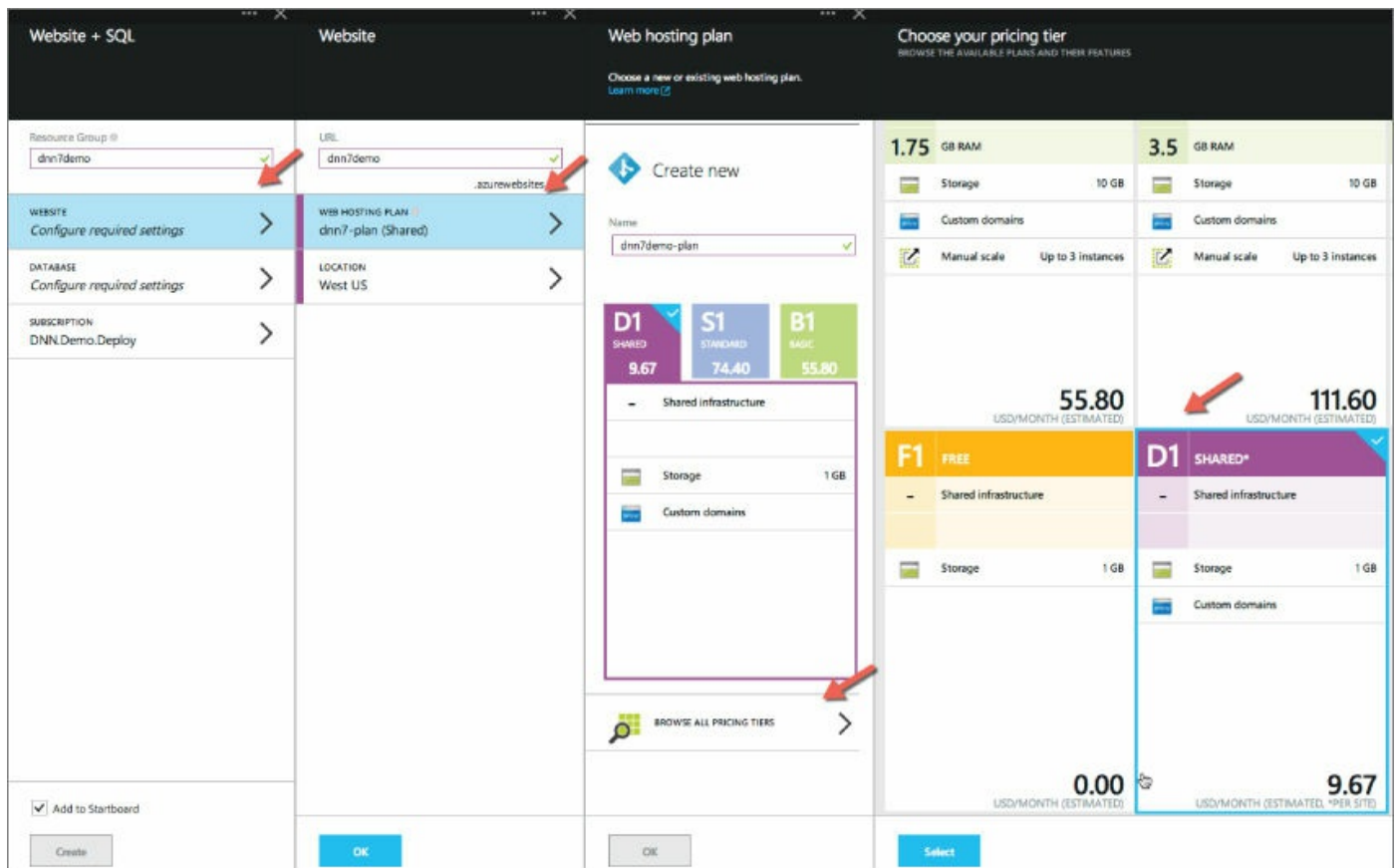


Figure 23.4

At this point you need to choose the actual plan you're going to use for your site. If this is a test site that you're creating for investigation, you can choose a Free site. Free sites are extremely limited, so for anything except a casual look, they don't work. Scroll to the bottom of the Web Hosting Plan blade and click **Browse All Pricing Tiers** to see the full list. The offers change over time,

but there are essentially four basic layers:

- **Free:** Shared infrastructure, limited processor allocation, and 1GB storage. Cannot assign a custom domain (works only on the *.azurewebsites.net domain)
- **Shared:** Shared infrastructure, limited processor allocation, and 1GB storage, but you can assign your own domain. You can use this for very small sites with low amounts of traffic and storage requirements.
- **Basic:** Dedicated infrastructure, increased storage and custom domains, and the ability to manually scale out the site up to three instances. Suitable for many production sites, and you can choose between different processing power levels.
- **Standard:** Like Basic, but includes higher scaling limits, plus the ability to use Autoscale, SSL certificates, backups, and the ability to create staging sites. Suitable for all production sites of all power levels (scaling to 10x4 core instances gives you a very powerful platform to run DNN on, if you have to deliver serious amounts of page views).

This example uses a D1 Shared infrastructure, as it is a suitable starting point for many people. The plan can be changed with a live site, so your choice isn't crucial at this point.

Once you have selected the plan, you need to specify where in the world you want your website to be located. Choosing this can be as simple as choosing the region that is closest to your intended customers. Choose carefully as moving resources can be difficult.

When you have made your choice on plan and geographic region, click the OK buttons from right to left. Each blade will close as the selection is made. You have locked in the Website resource.

Step 4: Creating the Database Resource

After you've chosen the website plan and named the website, the next step is choosing a database. Click the Database resource and then click Create a New Database.

The New Database blade will open and substitute in a database name based on the name of your website. This example sticks with the current naming convention and uses `dnn7demo` as the DB name.

Again, you will need to choose the database resources tier. Azure databases are priced by DTU, which is an Azure-specific measurement, which essentially abstracts the memory, processor, and throughput into a single value. The higher the DTU, the more database power you get. Working out specifically how many DTUs you require can be a case of trial and error.

This example uses S0 for the purposes of a demo site. From experience, it is very marginal running a DNN site on S0 for production use, and we recommend going to a S1 for most production sites, and to an S2 if the site sees a decent amount of traffic. Running on Basic is almost impossible with DNN—the install process itself can consume the entire DTU allocation.

[Figure 23.5](#) shows the process of creating a new database, choosing a name, and selecting a pricing tier.

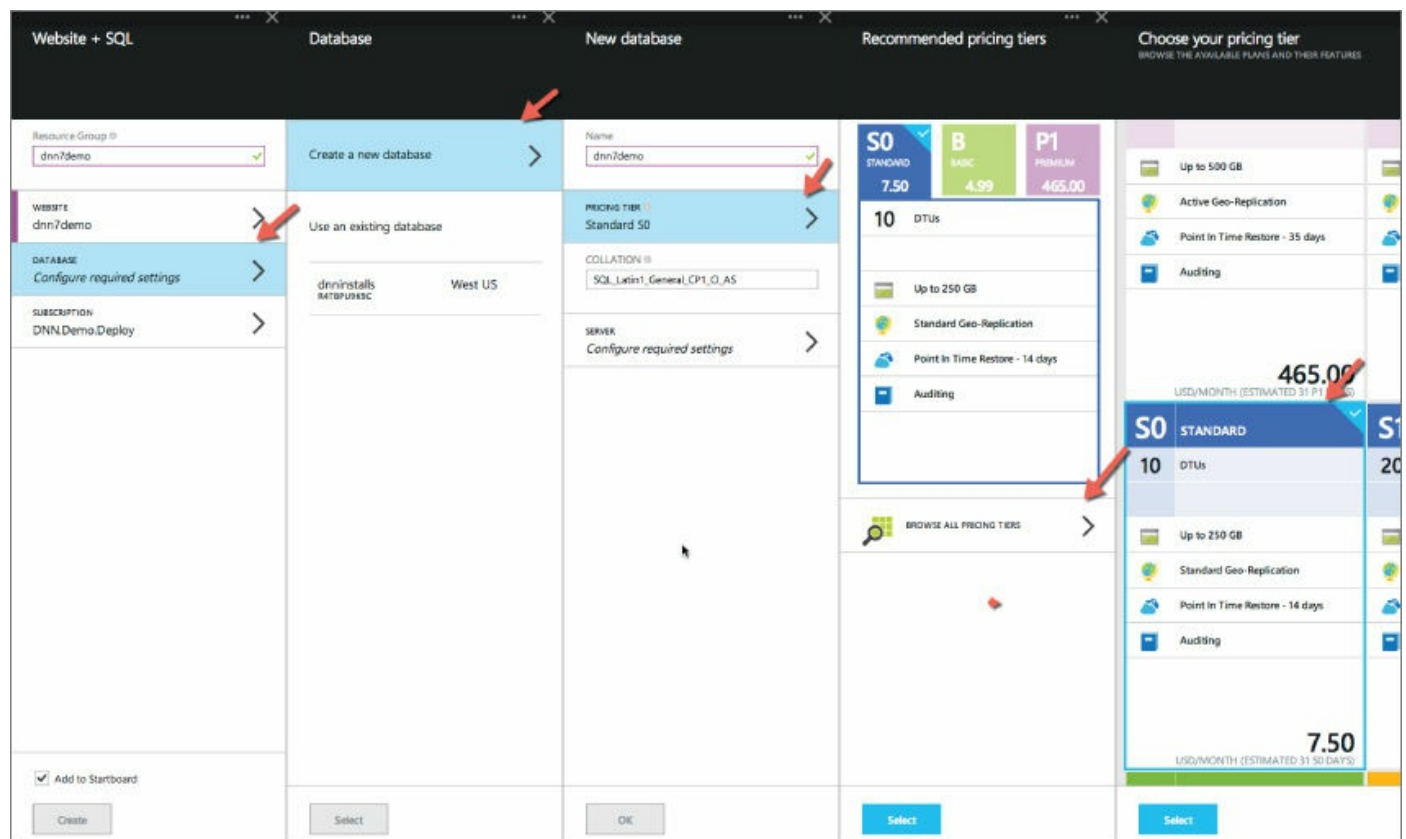


Figure 23.5

The prices can be a bit of a surprise compared to earlier Azure SQL pricing when Azure first released the product. Microsoft has changed direction slightly—the old plans were priced by storage, while the newer plans include a lot of storage but are priced instead on DTUs. The good news is that you can get much more consistent performance with the new plans. The result is that more power costs significantly more money.

Once the database name and plan is selected, the process is still not complete. You must select or create a database server where your database will reside.

Step 5: Creating a Database Server

Once the database is selected, it's time to select or create a server. If you have created a database previously, you may already have an Azure SQL Server created. If you haven't, you'll need to create one.

Despite the name, you're not actually creating a server of any kind. The Azure SQL service is an abstracted cloud service, and it runs on its own cloud of servers underneath. When you create a server, you're really just creating a container to put databases in, principally because that's a model people are familiar with and it helps with writing connection strings.

At this point we generally break again with naming convention and give the server a name. As it is a container, you should consider that you will probably create another database in it at some point—it's likely you'll get hooked on creating Azure Websites, and it's rare to stop at just one.

After choosing your server name, also choose the name of your server administrator account (which you can use to access all databases on the server) and set a password.

[Figure 23.6](#) shows the creation of the server during the database-creation process.

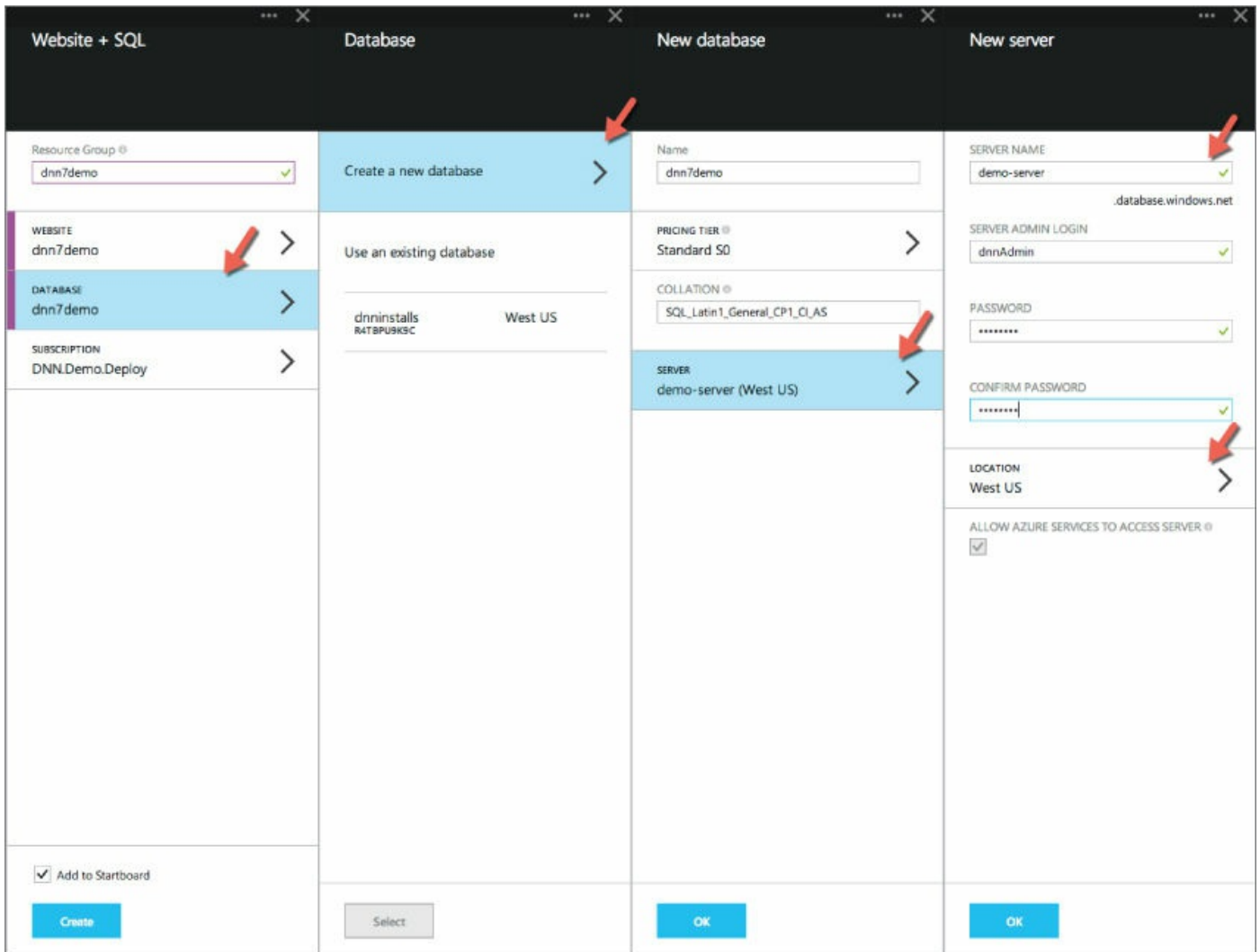


Figure 23.6

The final task when creating a server is to choose the location. It is very important that you choose the same location as the website (refer to the credentials file or the settings for the website). A DNN site that has the database in a different region than the website will run very badly.

After setting the database server name, admin login, and password, copy all these values into your credentials text file for safekeeping. You will need these credentials later.

Step 6: Creating the Azure Website

Once you've configured and selected the website and database resources, click the Create button to create the DNN site.

[Figure 23.7](#) shows a completed site-creation process, where the site database and subscription are chosen and the site is ready to be created.

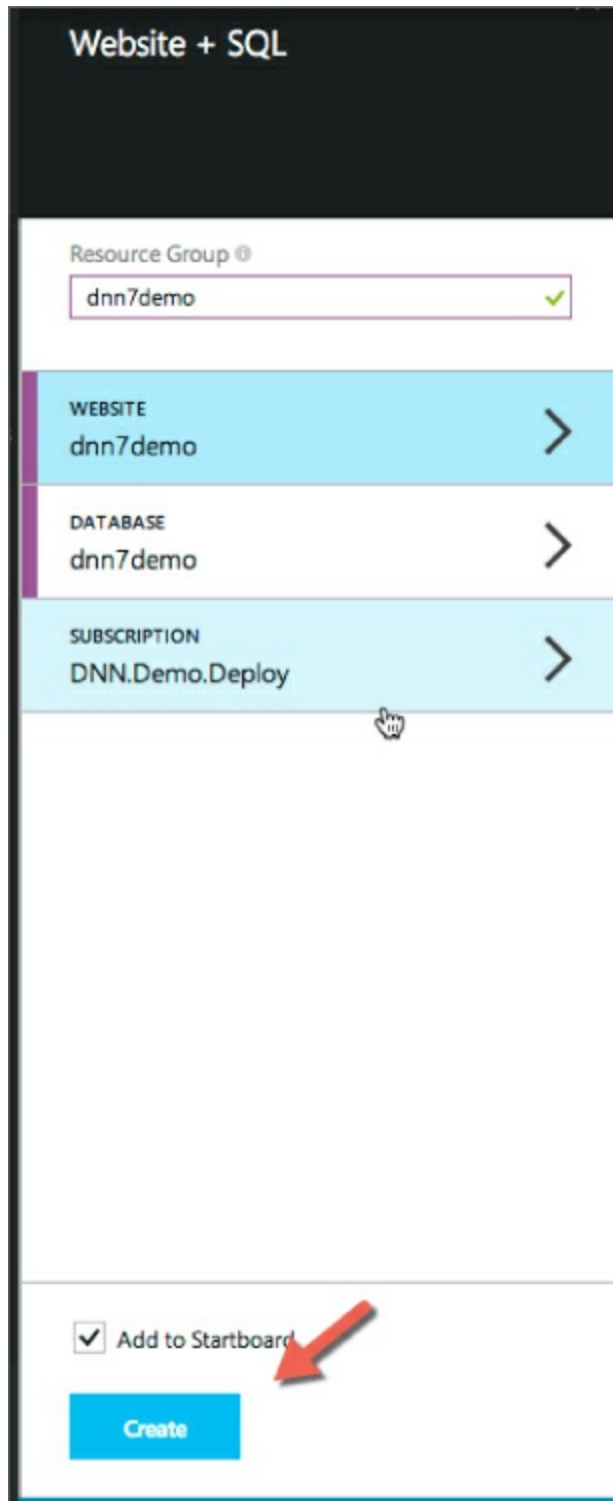


Figure 23.7

When you select DNN at the start of the process, Azure will create the necessary resources, load the latest DNN build, and start the install process. The open blades will close, and you'll see a status update as Microsoft Azure creates the various resources. [Figure 23.8](#) shows the status update for a website that's being created.

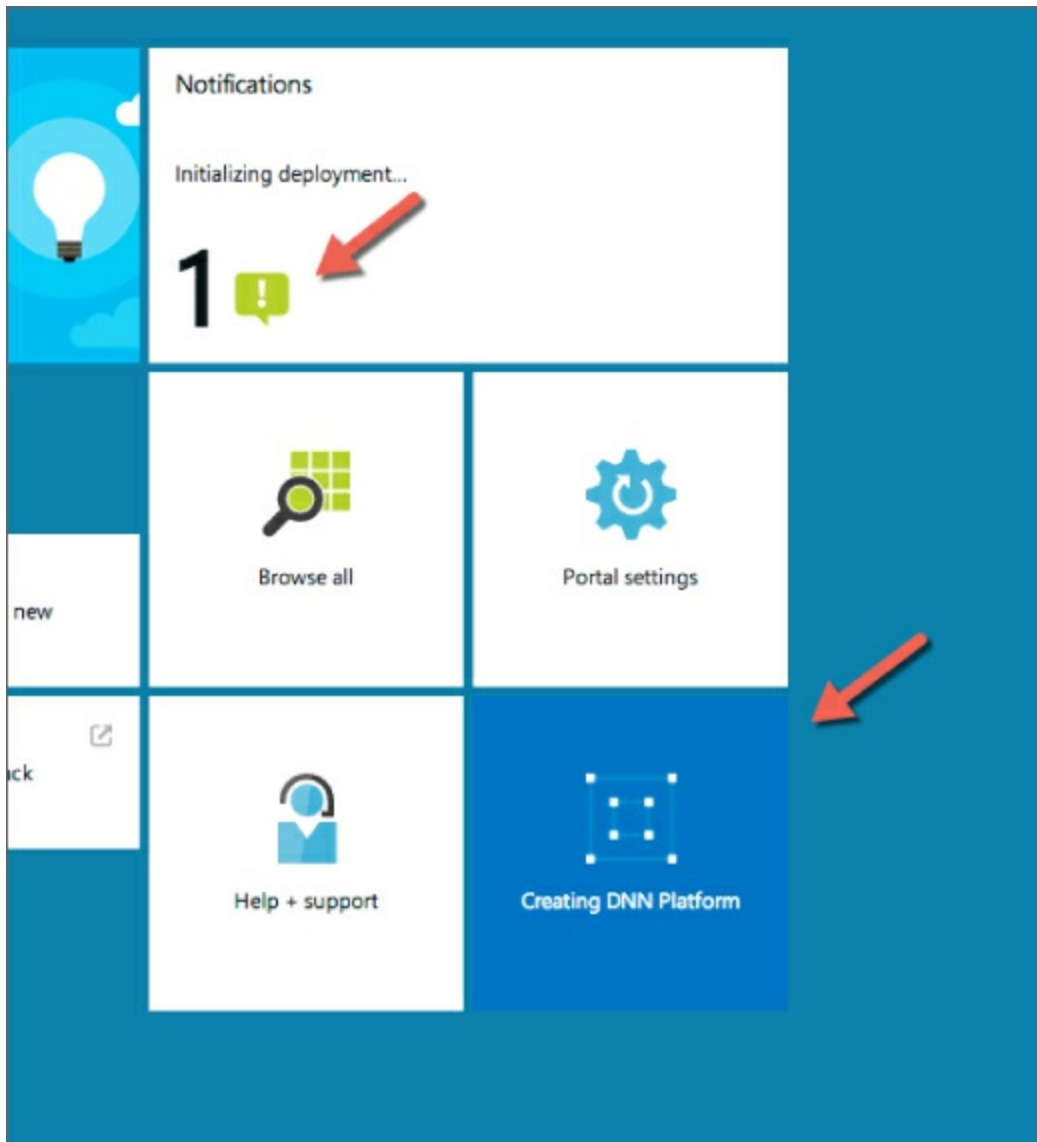


Figure 23.8

This process will take a short time and should be finished in about 10 minutes. The lower the tier of database you selected, the longer it will take. You can click the Notifications section to get a more in-depth description of the process.

When the site is complete, you'll get a notification. You can now click the site icon, which opens the blade for the Resource Group containing the site. The Resource Group is the complete container for the site and its database resources and will serve as the container for any future resources you create for this site. [Figure 23.9](#) shows the Resource Group blade and the website settings.

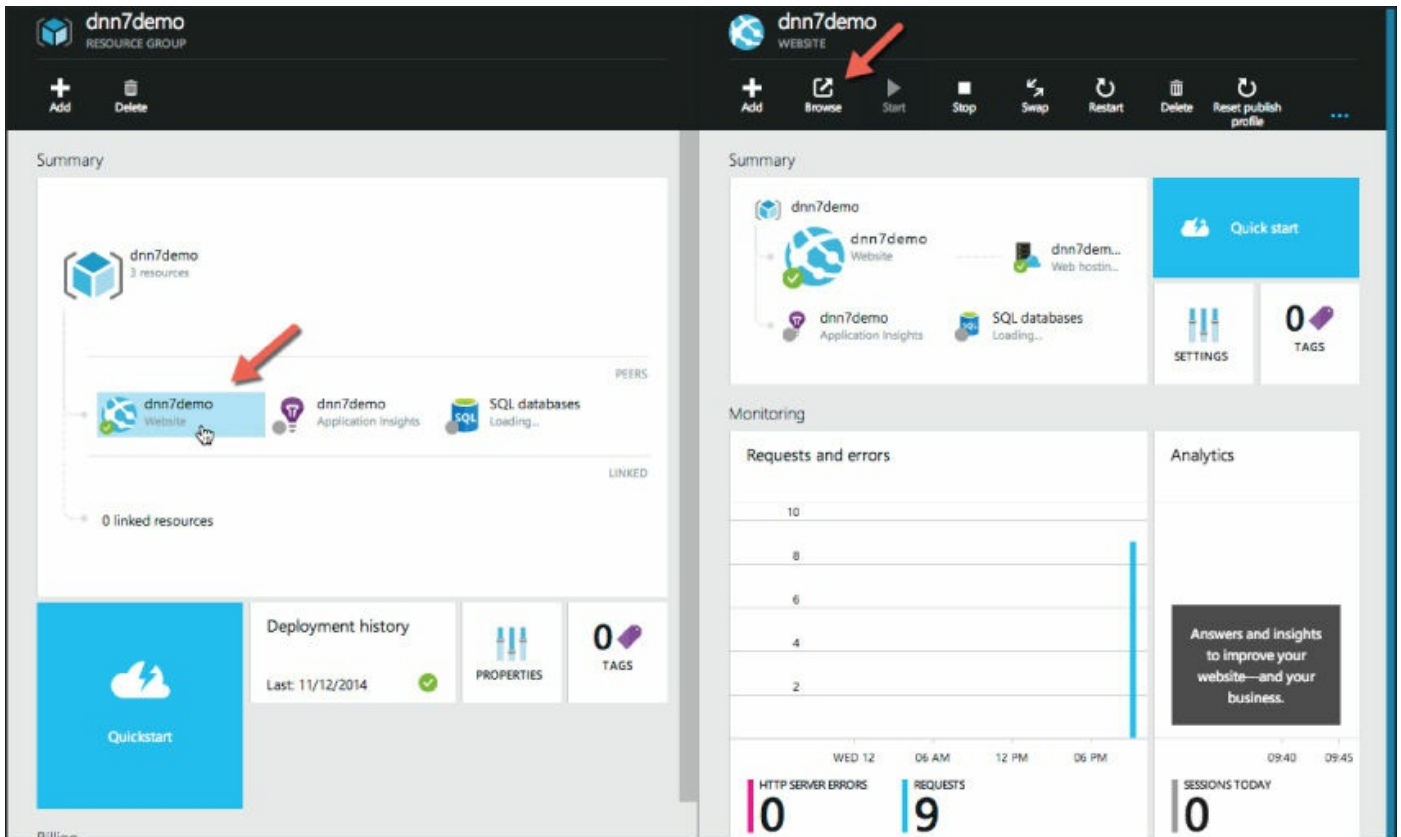


Figure 23.9

Click the Website icon to open the Website blade. This is the central point for configuration and settings of the site. Your DNN installation is not completed yet; you still need to visit the actual site. You can do this by clicking the Browse icon at the top of the blade; doing so will open the new site in a new browser window.

The first time the site runs it will take a while to start up. DNN contains dynamically compiled code, and the site will be sluggish until all these compiles are completed. The speed of this depends on the resources you selected when you configured the new site.

Once the site is started correctly, you'll see the DNN Installation Wizard screen.

Step 7: Running the DNN Install Wizard

To complete a DNN installation, you have to run the wizard. This is essentially no different than running the install wizard in any other environment, but be careful to enter all of the credentials correctly.

First, create the username and password for your host user. It's good practice

to use something other than the default value. Choose a strong password and a meaningful name for your website (this is the name of the first site created in the DNN install). Remember to update your credentials text file with the host user and password. [Figure 23.10](#) shows the DNN Install wizard with the host user and website installation values completed.

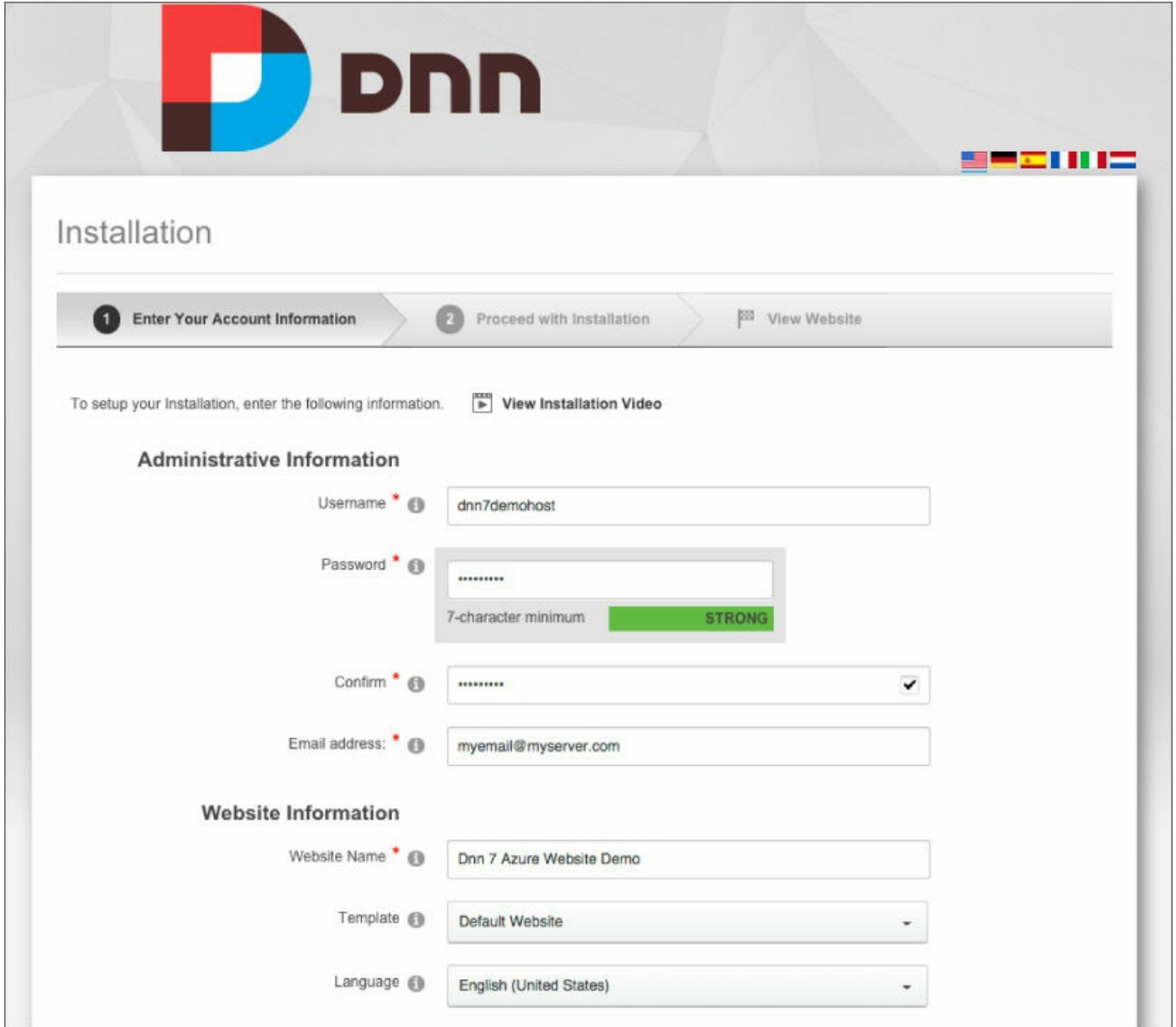


Figure 23.10

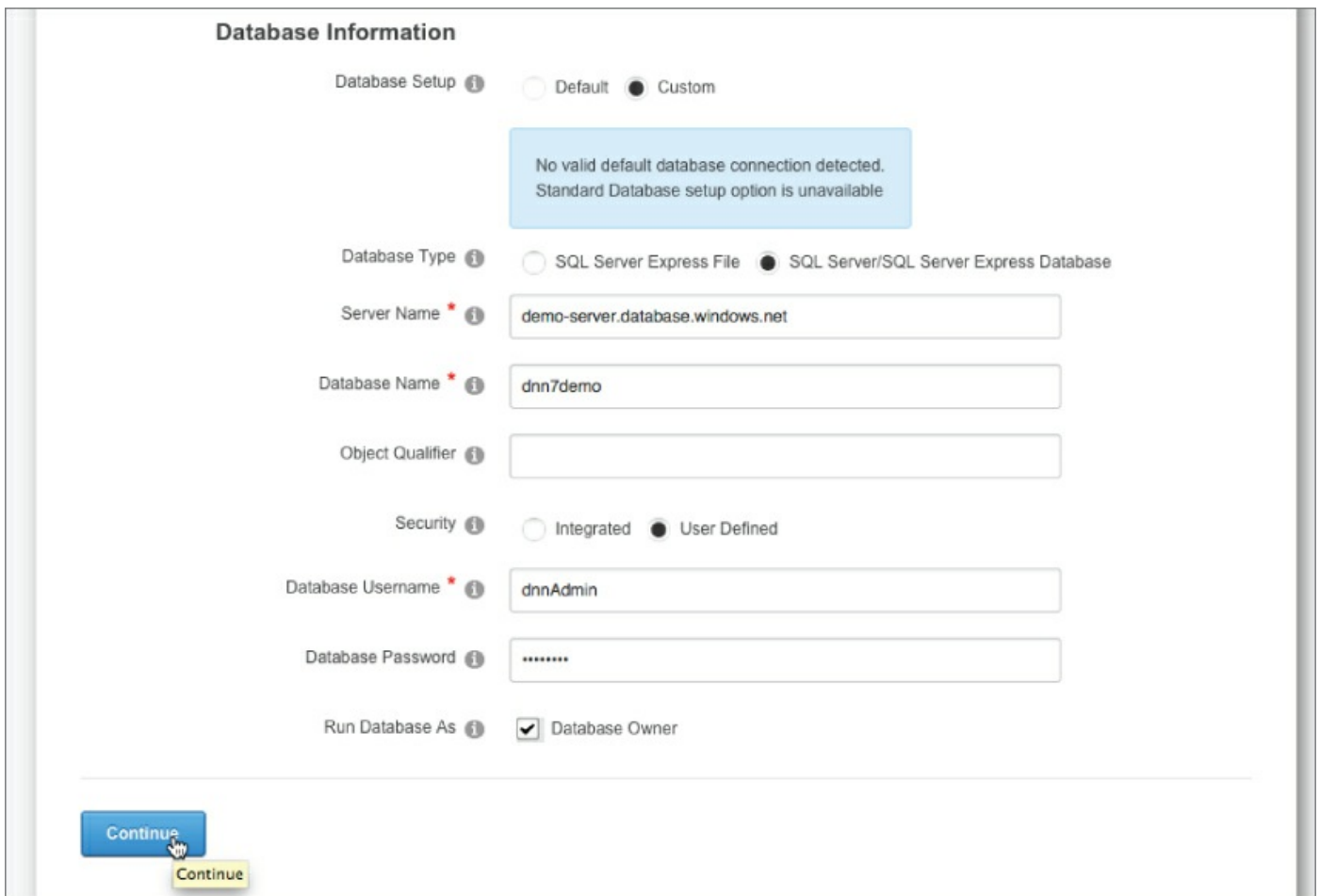
The database settings require careful input. Copy the values from your credentials text file to fill in the details. Choose the following options:

Database Setup: Custom

- **Database Type:** SQL Server/SQL Server Express database

- **Server Name:** {your Azure SQL Server name}.database.windows.net
- **Database Name:** {your database name}
- **Security:** User defined
- **Database Username:** {database admin name}
- **Database Password:** {database admin password}

[Figure 23.11](#) shows the Database Information portion of the DNN Install Wizard, with the example site credentials entered.



The screenshot displays the 'Database Information' section of the DNN Install Wizard. It features several configuration options and input fields:

- Database Setup:** Radio buttons for 'Default' (unselected) and 'Custom' (selected).
- Message:** A light blue box states: 'No valid default database connection detected. Standard Database setup option is unavailable'.
- Database Type:** Radio buttons for 'SQL Server Express File' (unselected) and 'SQL Server/SQL Server Express Database' (selected).
- Server Name:** Text input field containing 'demo-server.database.windows.net'.
- Database Name:** Text input field containing 'dnn7demo'.
- Object Qualifier:** Empty text input field.
- Security:** Radio buttons for 'Integrated' (unselected) and 'User Defined' (selected).
- Database Username:** Text input field containing 'dnnAdmin'.
- Database Password:** Password input field with masked characters '.....'.
- Run Database As:** A checked checkbox next to 'Database Owner'.

At the bottom left, there is a blue 'Continue' button with a mouse cursor hovering over it, and a yellow tooltip also labeled 'Continue' is visible below it.

[Figure 23.11](#)

Once you've entered the correct values, click Continue. The wizard checks the database connection and the install begins. [Figure 23.12](#) shows the Installation Wizard in progress.

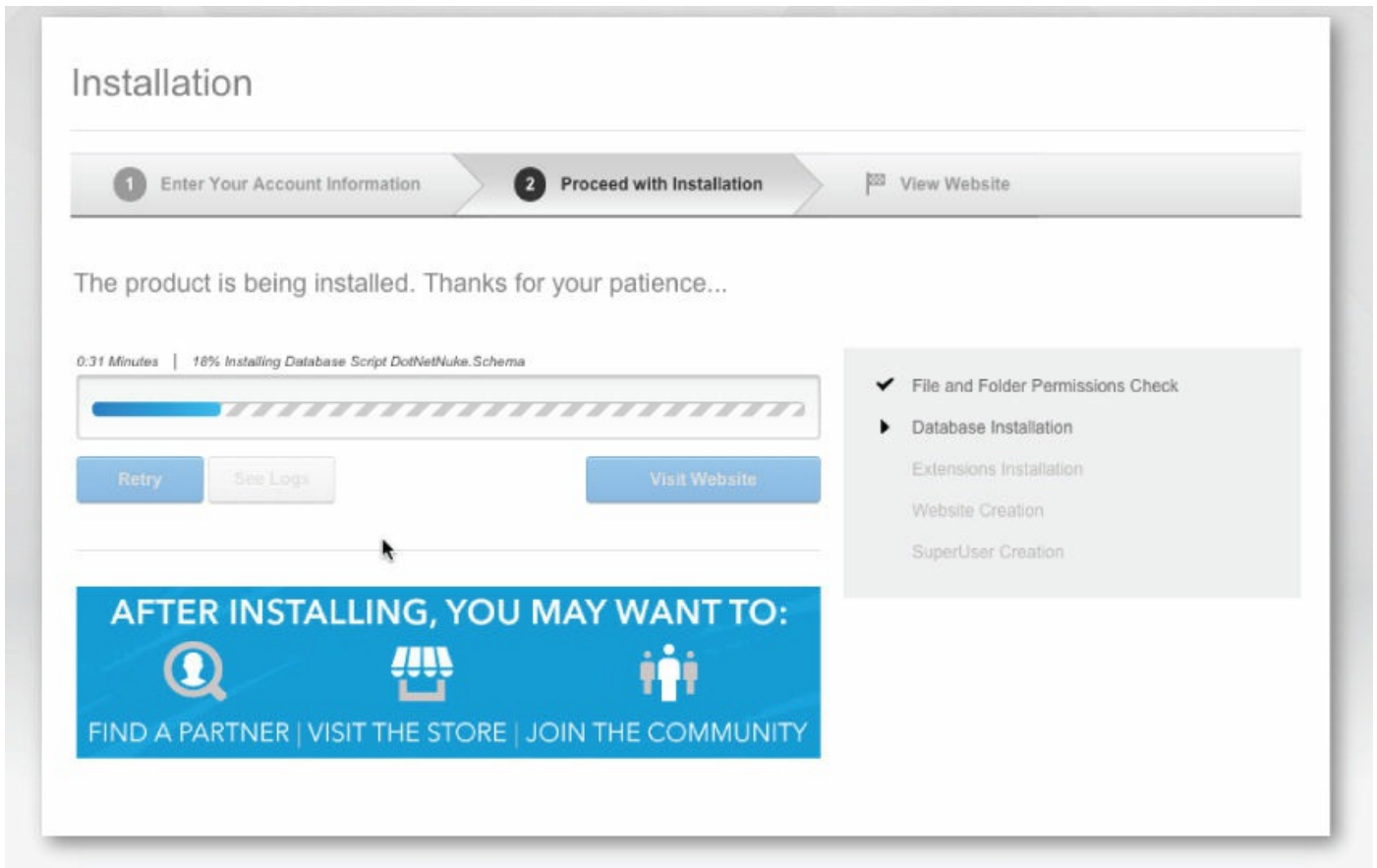


Figure 23.12

If you have installed DNN locally, you may find that the install process runs slower, again depending on your database and website tiers chosen during the creation of the Azure resources. The install scripts are DTU intensive, and lower specification database tiers take longer to process.

At the end of the process, you should have a completed website. [Figure 23.13](#) shows the completed install on Azure Websites.

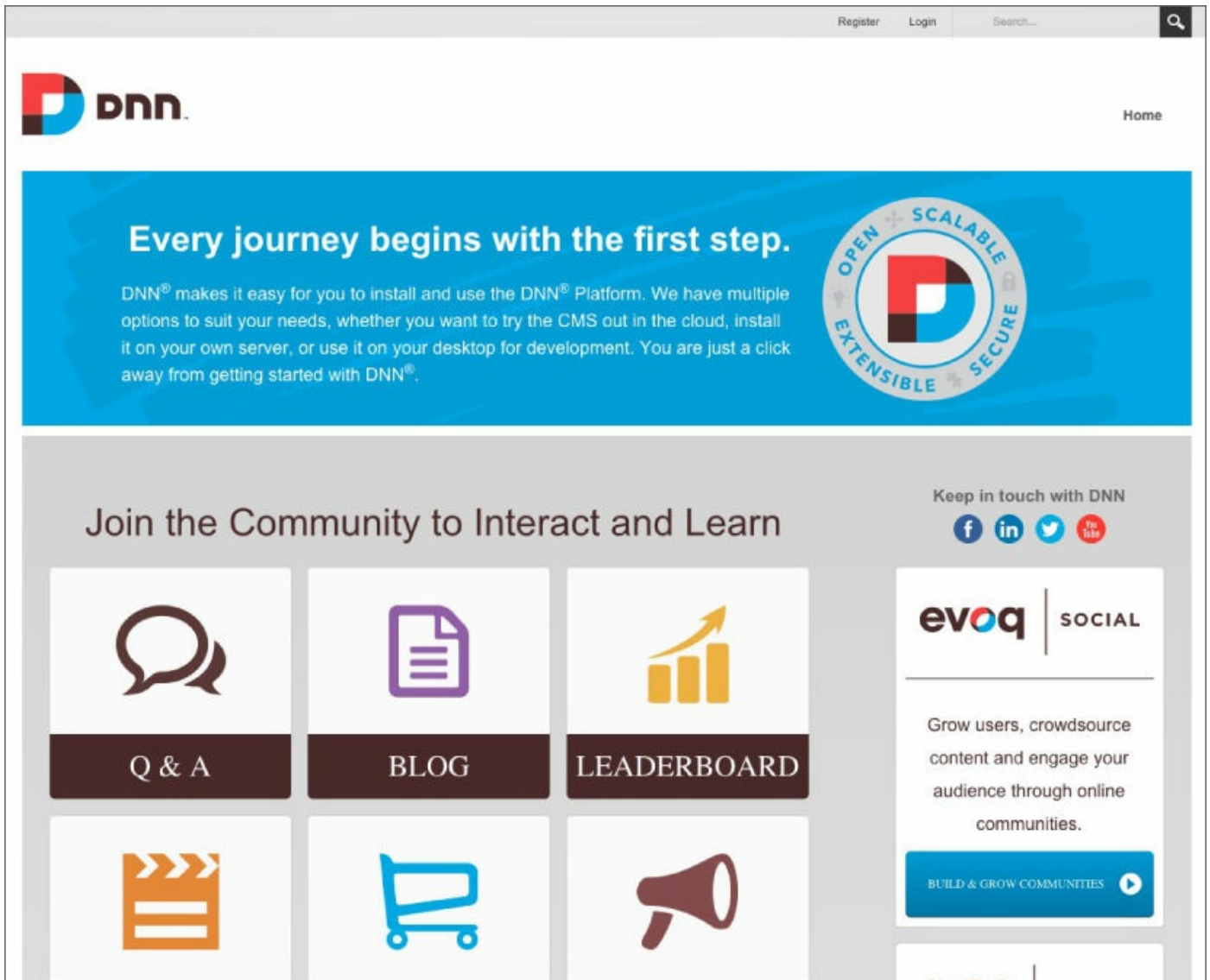


Figure 23.13

Now that you've created a site, you'll want to go through and set some configuration options to ensure it is working. The first step is configuring your domain name to point at the site.

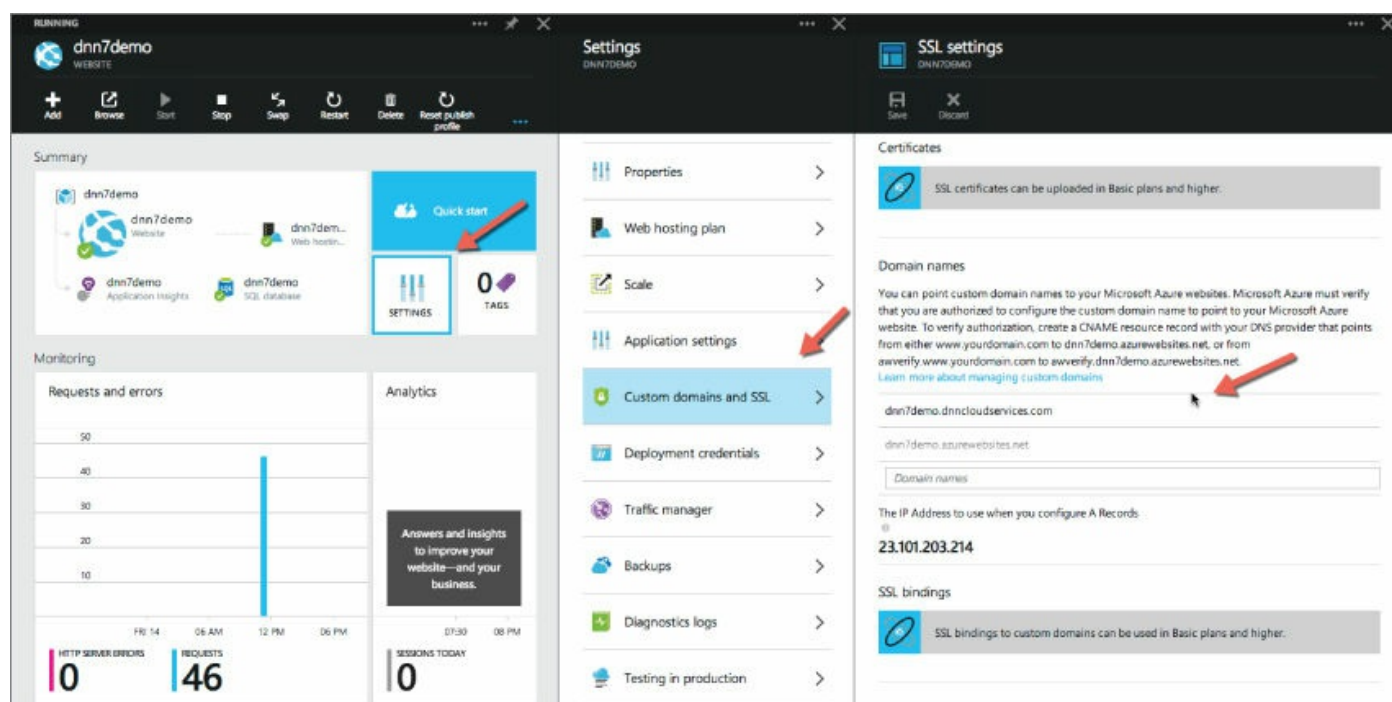
Step 8: Adding Domain Names

Your visitors will find your site(s) using the domain names you have. Pointing those domain name(s) at the Azure website requires some configuration at your DNS provider or Domain Name Registrar. The instructions for DNS configuration are provided by the Azure site and are simple to follow. Essentially, the domains work by using a `CNAME` record to point your www.example.com domain name to the Azure website URL. Alternatively, you can use an `A` record to point directly to the IP address. To

verify that you have authorization for this, you also need to create another CNAME record of `awverify.www.example.com` that points to `awverify.{name}.azurewebsites.net`.

Before embarking on this process, it's important to read and understand the documentation—on the Azure site and also on this Azure Documentation page: <http://azure.microsoft.com/en-us/documentation/articles/web-sites-custom-domain-name/>.

Once your DNS name is pointing at the Azure website, go to the Azure site, select the website, and click the Settings tile. In the Settings blade, click Custom Domains and SSL. [Figure 23.14](#) shows the Domains and SSL blade for the Azure website.



[Figure 23.14](#)

[Figure 23.14](#) shows the example `dnn7demo.dnncloudservices.com`, which is shared with a few other subdomains. Normally you would use a domain like `www.example.com` and don't have to use unusual subdomains.

Simply type the domain name you want to use in the list, and it will verify that the DNS entries are correct and then enable the Save icon. Click Save to commit the changes. This associates the domain name with the Azure website install.

If the DNS is not configured correctly, when you try to enter the domain name, you'll see the warning message shown in [Figure 23.15](#).

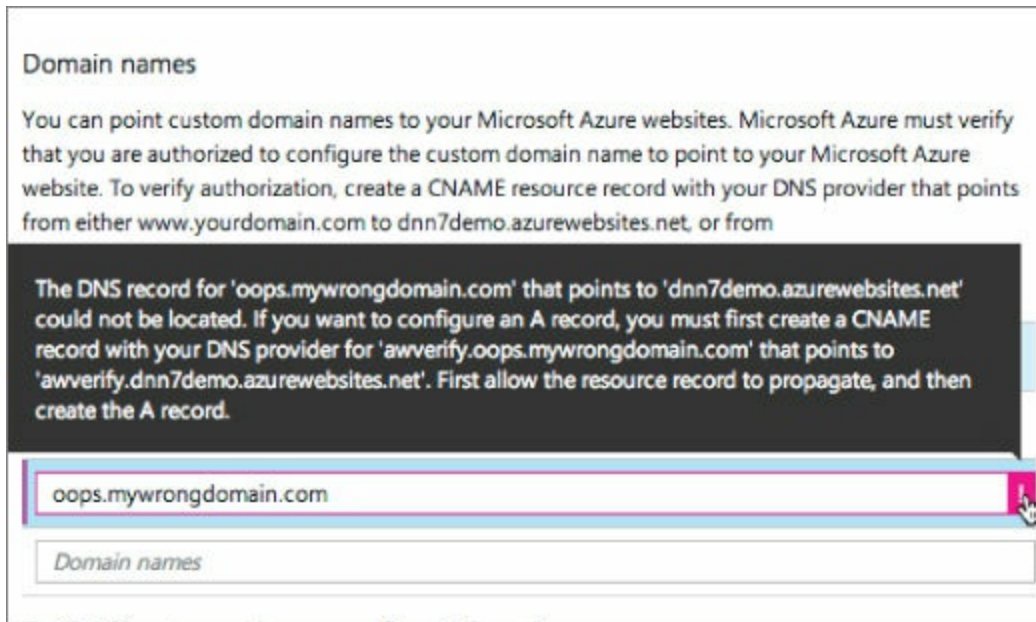



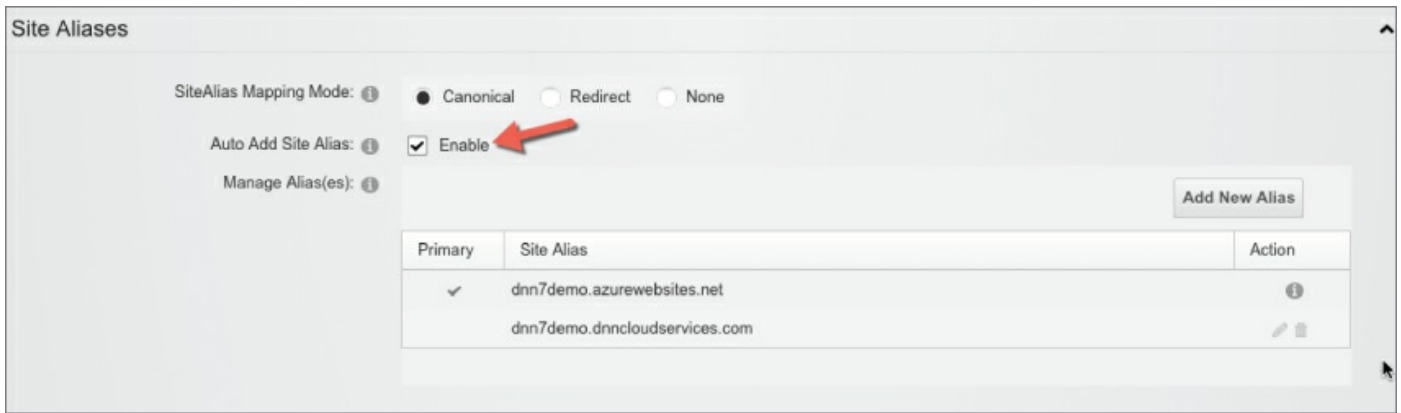
Figure 23.15

You may have made a mistake in entering the domain name, you might not have set it up correctly with your Registrar/DNS provider, or perhaps the change hasn't propagated across the Internet yet. The error message is long but informative. Go back over the steps and double-check. It's common to have to wait for propagation before the domain names work as expected.

Note that you can also associate the “naked” domain to the fixed IP address, but you still need to create the awverify.example.com CNAME entry to point to awverify.example.com so that the domain name is verified.

Once this is complete, click Save at the top of the blade, and then access your DNN site using the new domain name. The site should load using the new domain name. That is because DNN 7 is configured to use the Auto Add Site Aliases option by default. When a valid domain name requests the site, DNN automatically recognizes it and adds it to the list of valid site aliases.

Best practice dictates that as soon as you are finished adding domain names to the site, you should uncheck the Auto Add Site Alias option. Because Azure is a dynamic cloud environment, it's possible to have odd-looking URLs appear in the list of site aliases. It's not clear exactly where these come from—whether they are faults or intentional malicious actions. Extra entries in your Site Aliases table are unwanted, so disable the auto-add function as soon as your domains are configured correctly. [Figure 23.16](#) shows the Auto Add Site Alias checkbox in the Admin  Site Settings page of the newly installed site.



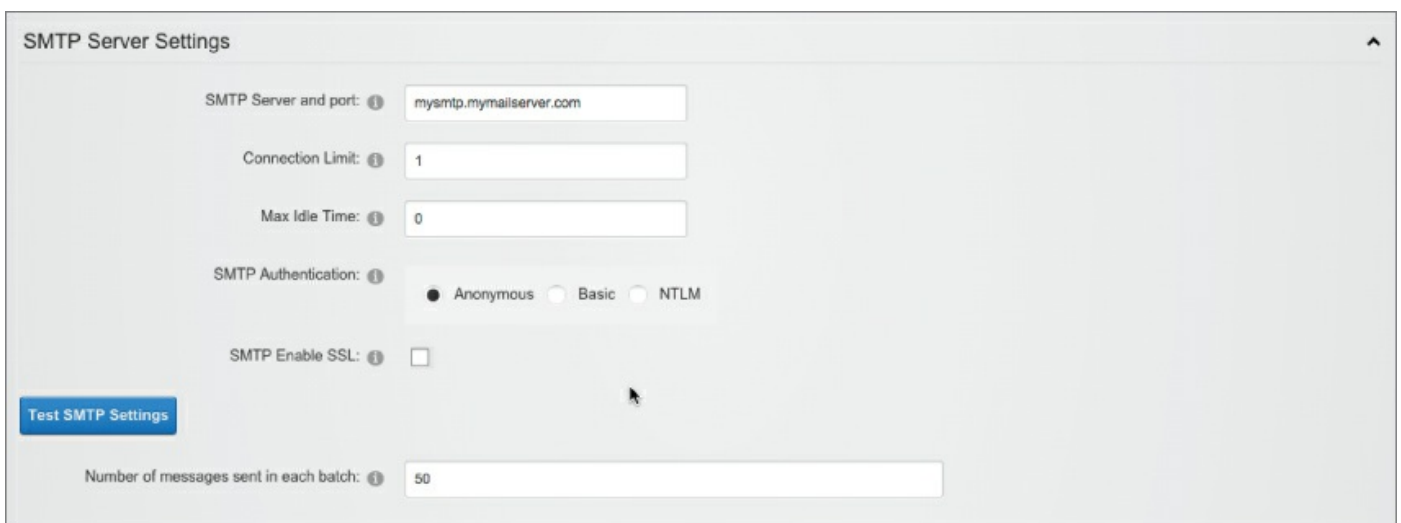
[Figure 23.16](#)

If the Auto-Add function is disabled or you want to install another site, follow normal practice and configure the DNS entry to point to the Azure website install. Then go into the DNN administrative interface and add the site alias manually.

Step 9: Configuring SMTP

All DNN installations require an SMTP server to work properly, whether to email customers or to send out password reset links. Azure is not a traditional hosting provider, so you don't get an email server to use with your Azure website. The first step in getting a working DNN site after the installation and domain configuration is to configure a working SMTP server.

SMTP configuration is done on the Host [Host Settings](#) [Advanced Settings](#) [SMTP](#) section. [Figure 23.17](#) shows the SMTP settings with an example server.



[Figure 23.17](#)

There is nothing specific about setting up SMTP on a DNN install on Azure Websites—the process is identical to any other DNN installation. You can use your existing SMTP server, which serves your company or personal email, or you can investigate one of the cloud-based SMTP providers. The only choice you don't have is leveraging Azure Websites itself to send mail; that simply isn't an option.

Step 10: Site Inspection

At the end of this process, you will have a new DNN 7 installation running on Microsoft Azure. Best practice is to systematically move through the site, checking on basic administration and editing functions, to verify everything works as expected. When everything checks out, you can immediately start creating content, installing third-party modules and working on the design of the site.

Remote Connections to Azure Websites

Once your new site is built and ready, you may want to start adding new content in bulk. There are multiple ways to connect remotely to an Azure website. Unlike most other hosting methods, obtaining an RDP connection is not one of the possibilities.

The three main possibilities for connecting remotely to an Azure website are

- FTP
- Web Deploy (WebMatrix)
- Git

The Git connections are designed to allow the owner of a site to push changes to their site after committing in Git. This type of continuous deployment model suits custom-built websites but isn't a great fit for DNN, which is designed to be extended by installing extensions rather than directly modifying source code.

Web Deploy is a Microsoft technology for allowing simple connection and synchronization with a local copy of the site using Visual Studio or WebMatrix. This works well but is a little complicated to set up. If you're comfortable with this method, it may work for you.

In this example, the connection is going to be demonstrated using FTP. This is a technology most people are familiar with and is a robust and dependable way of transferring data to and from websites.

Remotely Accessing Azure Websites with FTP

Accessing an Azure website with FTP is very similar to using FTP with any other website. You'll need access to the Azure site and good-quality FTP software. It's not recommended to use a basic FTP program such as browsing with Windows Explorer.

Step 1: Creating FTP Credentials

The first step in gaining FTP access is to create the credentials used to connect. Go from your Website blade, click Settings, and, in the Settings blade, click Deployment Credentials.

You can either accept the default username created by Azure or create your

own. Enter a strong password and click the Save icon. [Figure 23.18](#) shows the creation of FTP credentials in the Deployment Credentials blade.

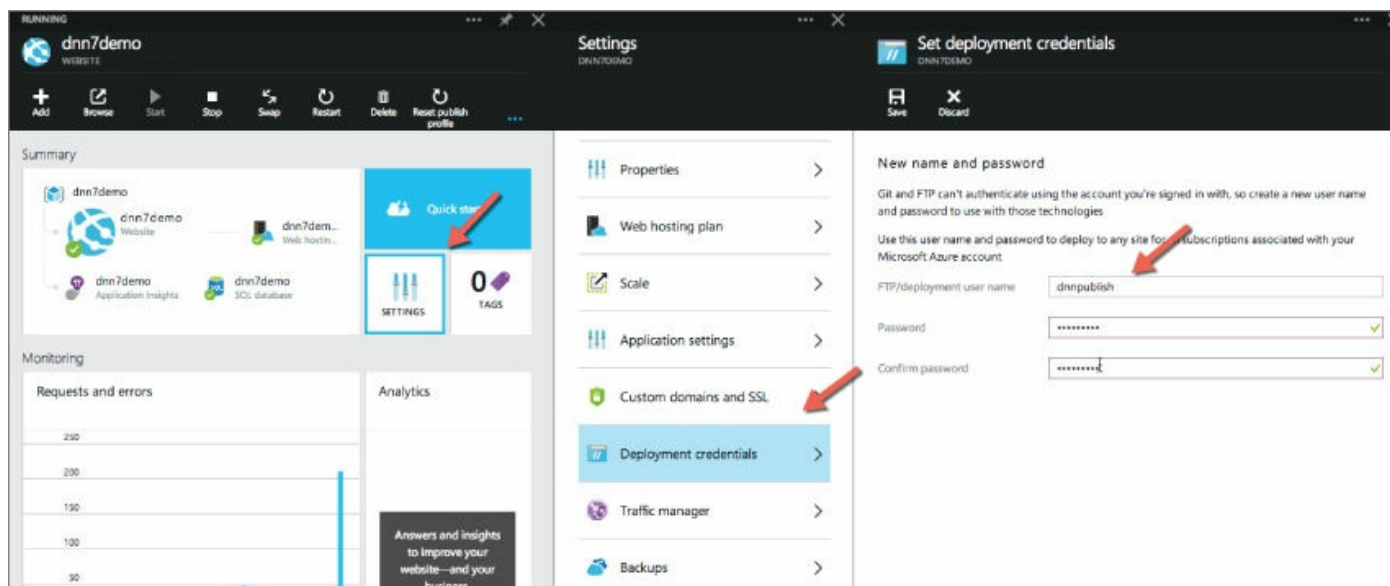


Figure 23.18

Take a moment to copy the FTP username and password to your credentials text file so you have a record.

Step 2: Obtaining the FTP Endpoint

Once the credentials are set, click Properties to open the Properties blade. This provides you with a number of FTP endpoints you can connect to.

- FTP Deployment User is the qualified name of the user you just created.
- FTP Host Name gives you the endpoint for the root of your site.
- FTP Diagnostic Logs gives you the endpoint for the diagnostic logs of your site.

[Figure 23.19](#) shows the FTP endpoints in the Website Properties blade.

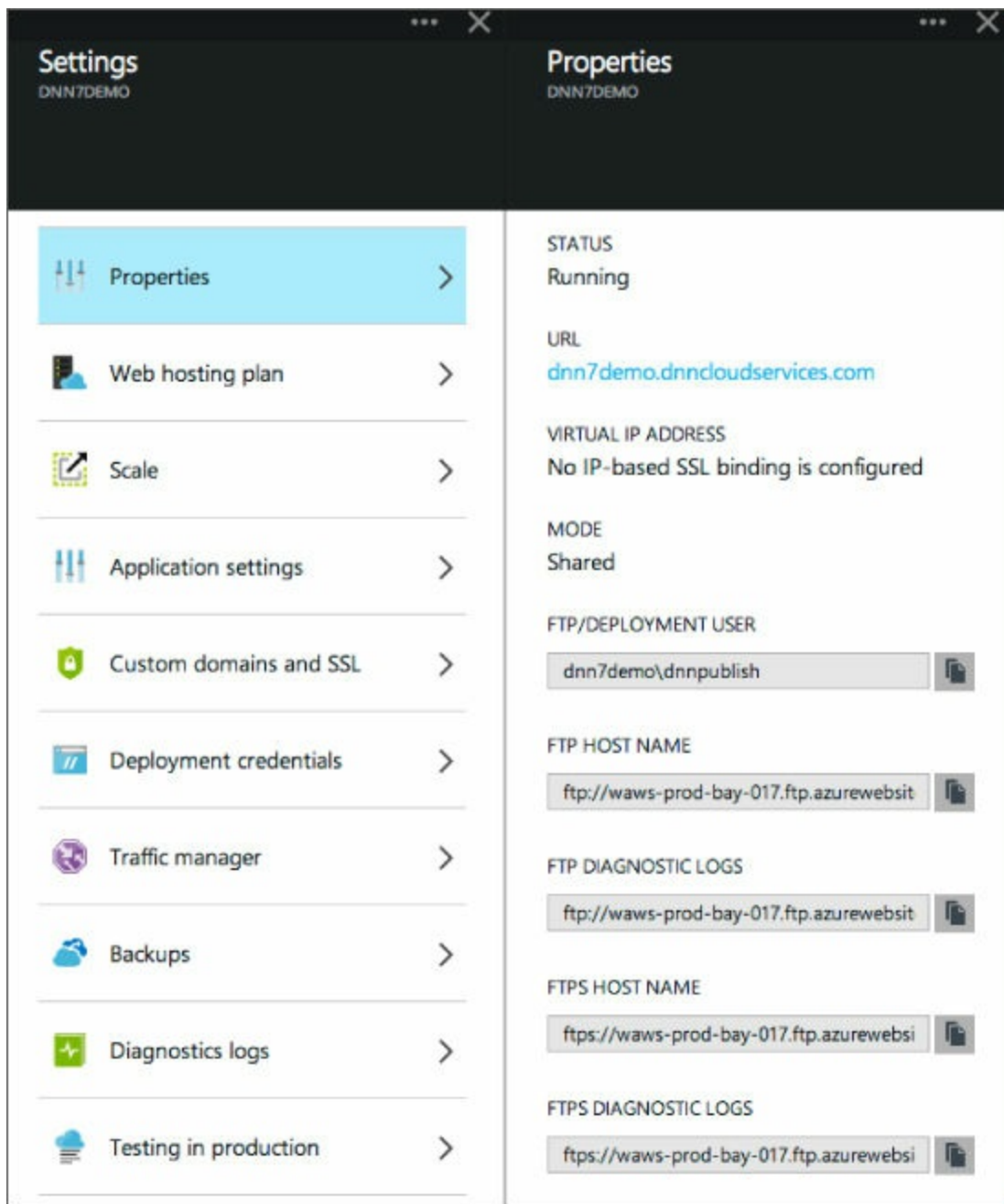


Figure 23.19

These are repeated for the FTPS versions. Copy the FTP host name into your credentials file as your FTP endpoint.

Step 3: Opening the FTP Connection

To connect, open your favorite FTP program and enter the credentials. [Figure 23.20](#) shows a connection to the example site using FileZilla, a popular open source FTP client.

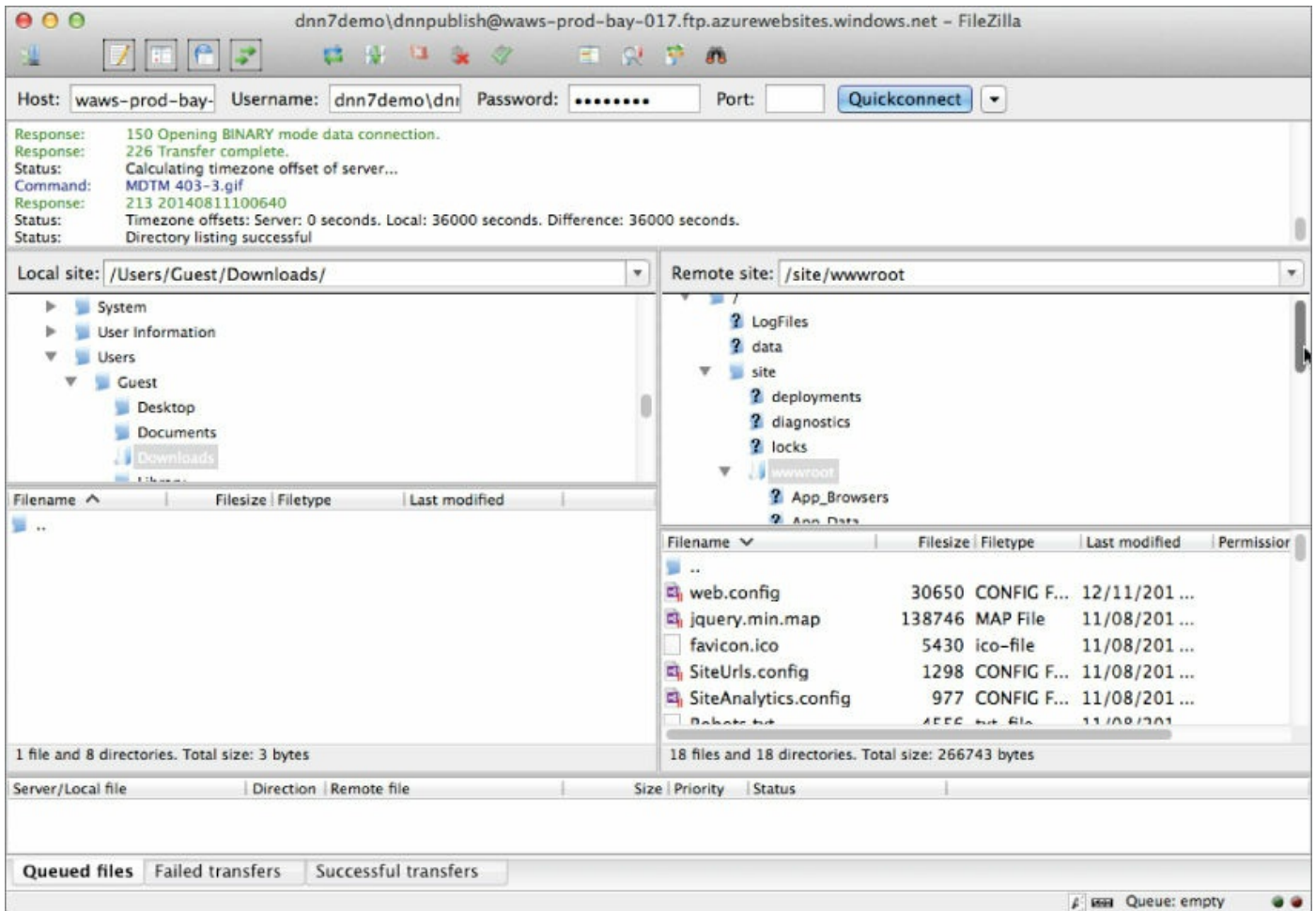


Figure 23.20

You can use the Copy icon on the endpoints to copy the contents into your clipboard and paste them into your FTP program. Alternatively, copy them from your credentials file. Then connect and you'll see the structure of the Azure website.

Once you're connected, you can upload/download code/content for the site or access any diagnostics and weblogs. The site is stored in the `/site/wwwroot/` path.

Once you have completed this process, you can move content and upload it to your Azure website as needed.

Backing Up Your Azure Website

Now that the site is installed and configured, it's time to take a full backup of the site. A DNN installation is a combination of the application files (content, code, and configuration files) and the database. A correct backup of any DNN installation is a combination of a backup copy of the database and a backup copy of the application files. Because there are dependencies between the database and the application files, any valid backup should be a snapshot of both at the same date/time.

Creating a Combined Application/Database Backup

The following process describes how to create a combined database/application files backup using the tools available in Azure Websites.

Step 1: Choosing the Correct Website Pricing Plan

Azure Websites provides built-in backup capability for performing site and database backups, but this is available only in the Standard plans. To demonstrate, the `dnn7demo` site will be switched from the Shared plan it was initially configured with to a Standard plan.

[Figure 23.21](#) shows how to select a new Pricing Tier blade from within the Website blade.

The screenshot shows the Azure portal interface for a website named 'dnn7demo'. The top navigation bar includes 'Add', 'Browse', 'Start', 'Stop', 'Swap', 'Restart', 'Delete', and 'Reset publish profile'. The main content area is divided into several sections:

- Usage:**
 - File System Storage:** DNN7DEMO-PLAN, 1.43%
 - Quotas:** DNN7DEMO-PLAN, Memory Percentage 57%, CPU Percentage 3%
 - Scale:** DNN7DEMO-PLAN, Autoscale Off, Instances 1, Sites 1
 - Pricing tier:** DNN7DEMO-PLAN, 1 Small Instances, BASIC FEATURES (represented by icons for storage, website, auto-scale, backup, staging, and traffic manager).
- Estimated spend:** DNN7DEMO-PLAN, with a note: 'Billing data will arrive as you accrue costs'.
- Operations:**
 - Events in the past week:** DNN7DEMO, bar chart showing 15, 10, and 5 events.
 - Alert rules:** DNN7DEMO, 2 rules (indicated by a green checkmark).
 - Streaming logs:** Enabled.
 - CONSOLE:** Available.
 - PROCESSES:** Available.
 - Backups:** Section visible at the bottom.
 - WebJobs:** Section visible at the bottom.
- Choose your pricing tier:** A dialog box showing a table of plans:

Plan	Core	GB RAM	Storage	SNI/IP	Instances	Backup	Staging	Traffic Manager	Estimated Price (USD/MONTH)
S1 STANDARD	1	1.75	50 GB	5 SNI, 1 IP	Up to 10	Daily	5 slots	Geo availability	74.40
B1 BASIC	1	1.75	10 GB	Custom domains	Up to 3				

Figure 23.21


Once you select a new plan, the website is switched to a new plan, which takes a few moments. Once it is finished, you'll see items like the File System Storage quota change.

Step 2: Linking the Database and File System

Before you can get Azure Websites to take a backup, you have to tell the website what the linked database is so that it can combine the application files and database.

Open the Website Settings Application Settings blade. Find the Connection Strings setting. Even though the DNN installation includes a connection string in the `web.config` file, this is not automatically added to the website

settings. Add the `SiteSqlServer` connection string for your installation by opening the `web.config` file and searching for `SiteSqlServer`.

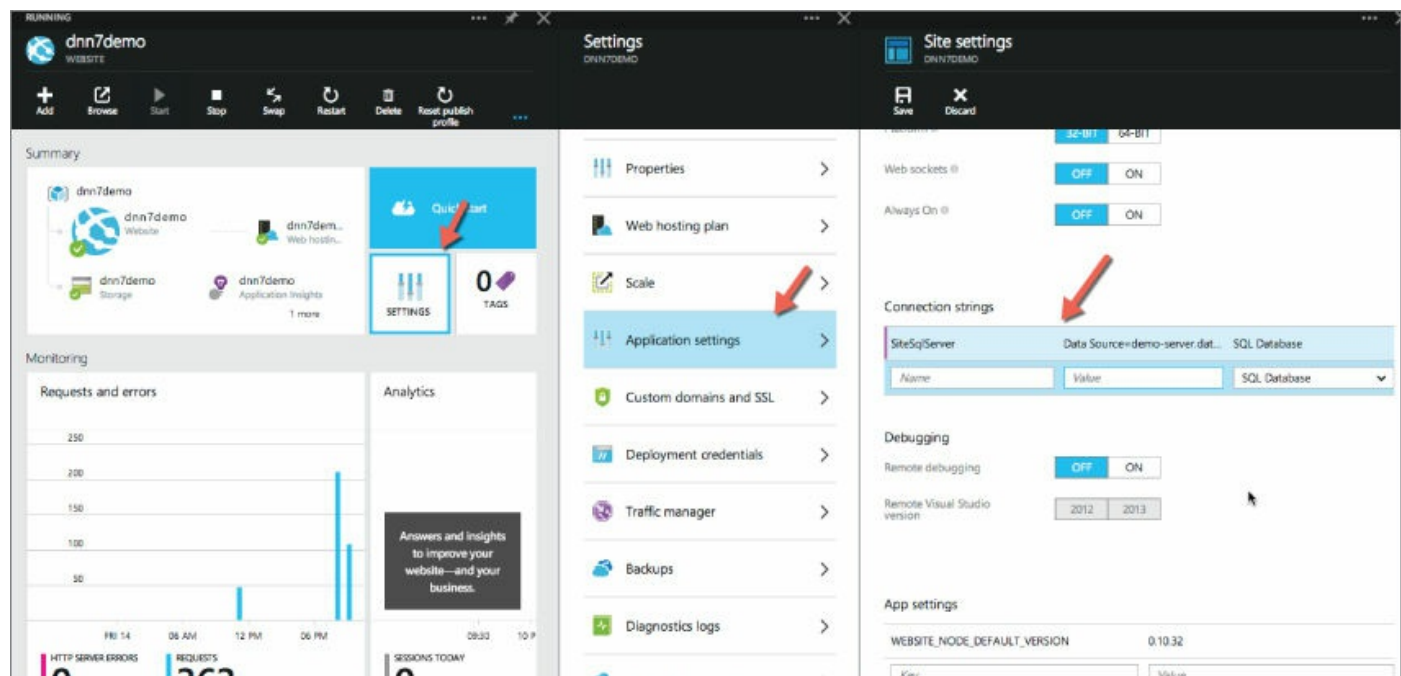
If you don't yet have a copy of the `web.config` file, you can open it in the site using Host  Configuration Files, or you can use FTP to download a copy as described in the previous section.

In the Connection Strings section, add `SiteSqlServer` in the Name box and copy/paste the connection string as contained in quotes in your `web.config` file, into the Value box. Ensure that SQL Database is selected as the type and use the Save button at the top of the blade to save the changes. [Figure 23.22](#) shows the location of the connection string in the `web.config` file.

```
<connectionStrings>
<!-- Connection String for SQL Server 2008/2012 Express -->
<add name="SiteSqlServer" connectionString="Data Source=demo-server.database.windows.net;Initial Catalog=dnn7demo;User ID=dnnAdmin;Password=<password>" />
<!-- Connection String for SQL Server 2008/2012
<add name="SiteSqlServer" connectionString="Server=(local);Database=DotNetNuke;uid=;pwd=;" providerName="System.Data.SqlClient" />
```

[Figure 23.22](#)

When you're finished, you should have a connection string entered for the site within the Site Settings blade. [Figure 23.23](#) shows the connection string correctly entered.



[Figure 23.23](#)

At the completion of this process, the database and file storage are linked together for the purpose of backups.

Step 3: Creating a Storage Account for Backups

The next requirement is to indicate where to store your backups. The backups are automatically sent to an Azure storage account. If you don't have a suitable Azure storage account, it's time to create one.

Click the + New control in the bottom-left of the Azure site and select Storage, cache + backup as the category (you may need to click the Everything link to see all of the categories). Then select Storage. [Figure 23.24](#) shows the selection of options to create the new storage account.

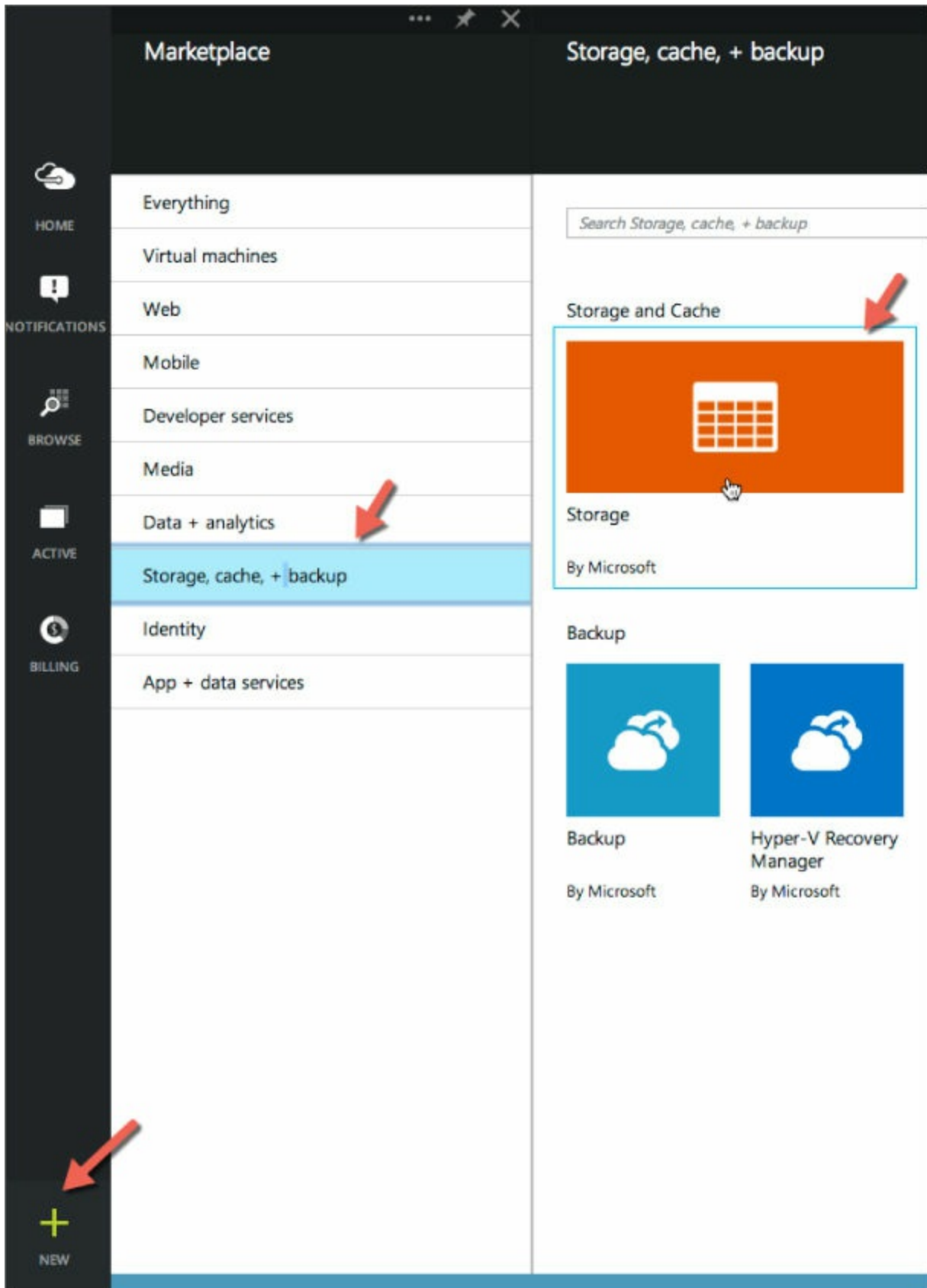


Figure 23.24

Enter the same name for the storage account, check to make sure it's going

into the same resource group as the website, and then select the Storage Account type. We recommend using the Geographically Redundant Storage (G1 GRS), which means a copy of your files are stored in two separate data centers in the same region, making loss of the files highly unlikely.

Again, ensure that you're creating the storage account in the same Azure region as where you created your website and database. This is very important for speed and to avoid outbound data charges. [Figure 23.25](#) shows the selection of the pricing tier, subscription, and region.

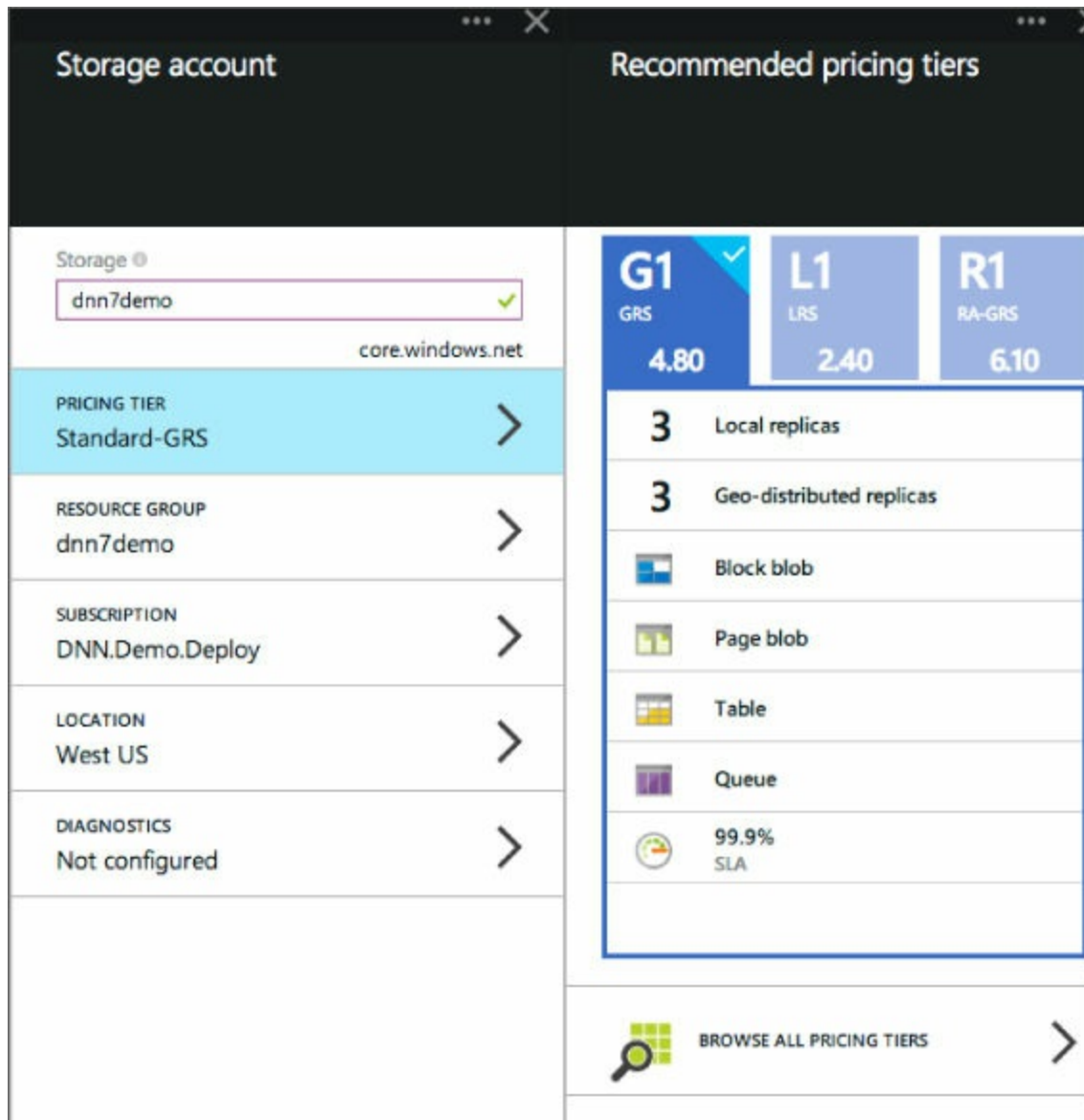
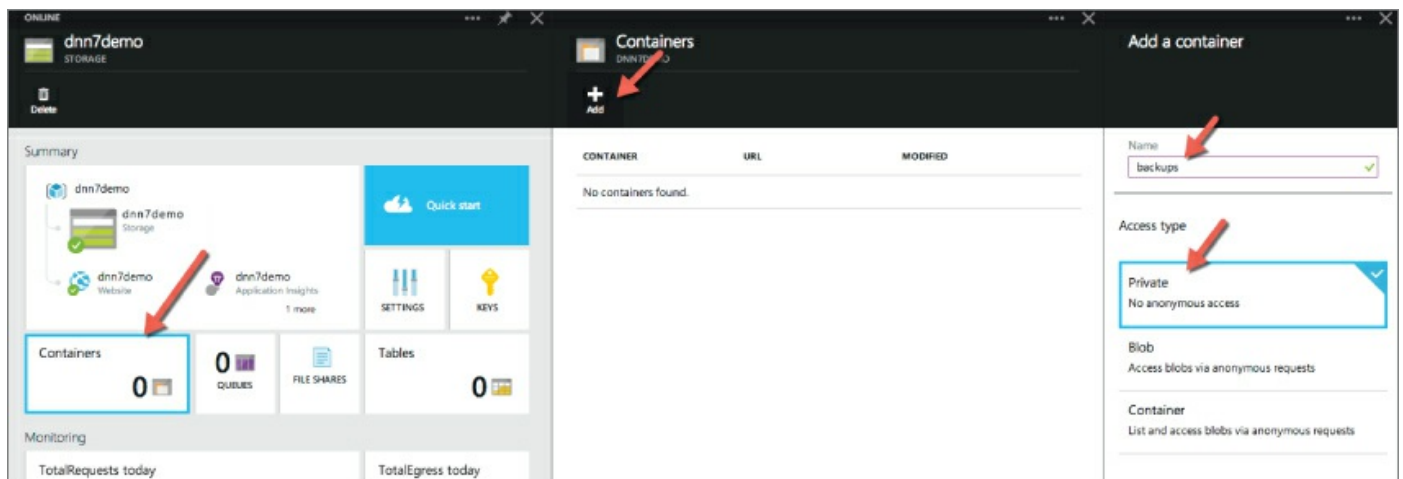


Figure 23.25

Click Create when you have everything ready. The storage account will take a few moments to create. When it's ready, click the storage account and open the Storage blade.

Step 4: Creating a Container in the Storage Account

A storage account is an empty bucket—it's not like a folder on an operating system, ready to accept files. Storage can be in the form of queues, containers, or table storage. Containers are the appropriate use for storing files like backups, so create a new container by clicking the Containers tile. When the Containers blade opens, use the + Add icon to add a new container. It's best practice to leave these containers set to private access so that anonymous requests can't download your backup files. [Figure 23.26](#) shows the options for creating a new container.

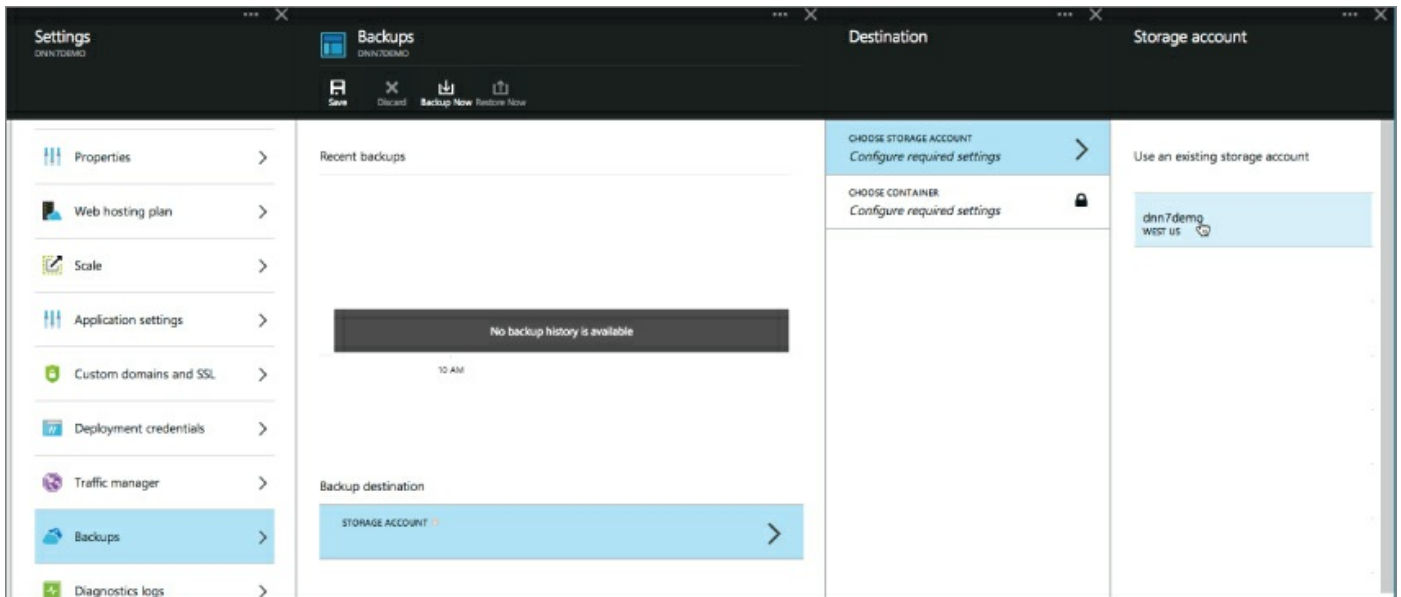


[Figure 23.26](#)

The container is then created and ready to accept backups.

Step 5: Creating the Backup

After you've created a storage account and container, return to the Website blade and open the Settings ➔ Backups blade. The storage account and container must be selected before the backup can be created. [Figure 23.27](#) shows the selected options for creating a backup on the created storage account and container.



[Figure 23.27](#)

To create the backup after the storage account and container have been selected, click **Backup Now** on the Backups blade. Make sure you have included the `siteSqlServer` database.

The backup should take a moment. [Figure 23.28](#) shows the list of completed backups with the new backup included.

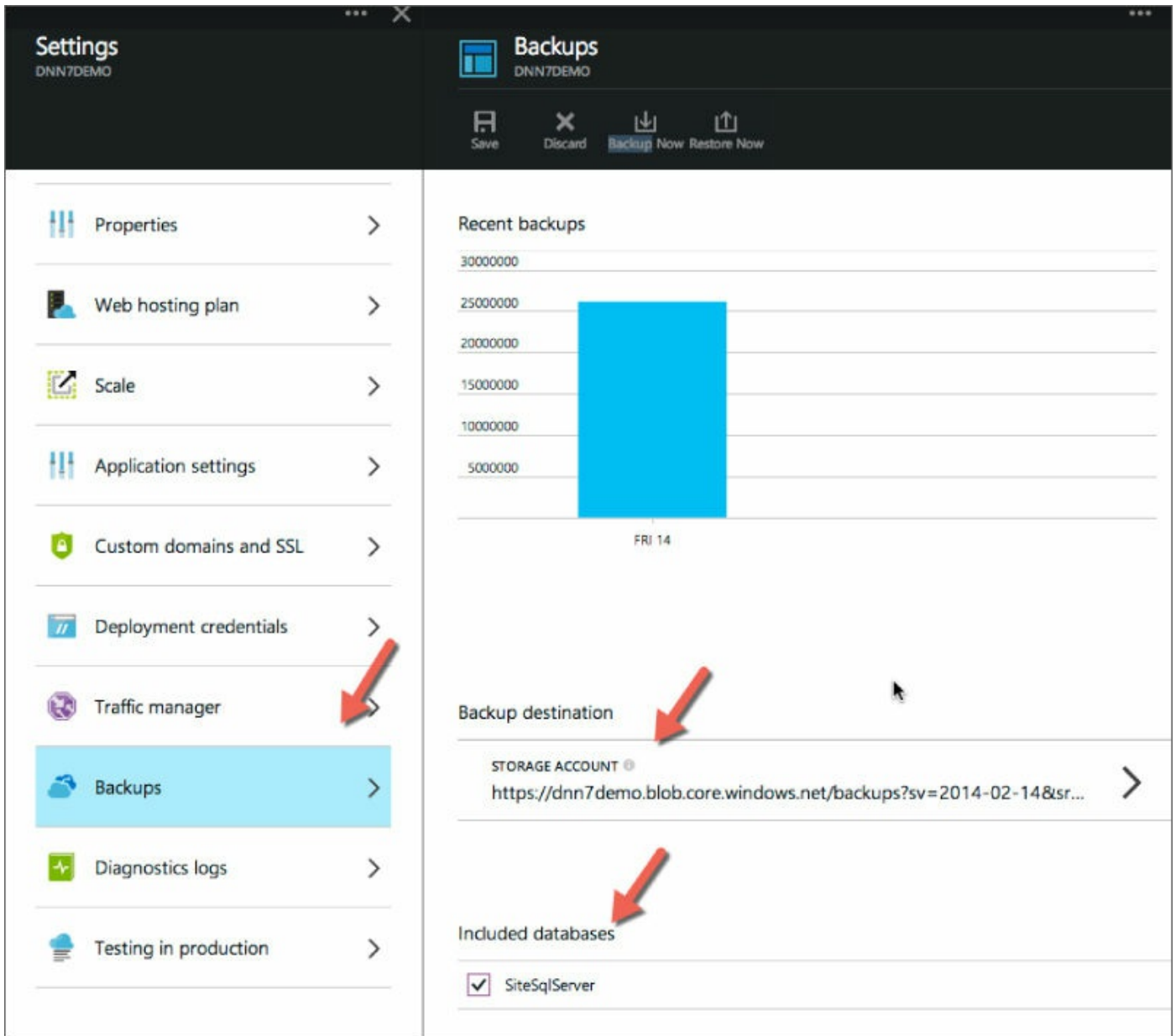


Figure 23.28


Step 6: Creating Scheduled Backups


When a successful backup is operating, it's a simple task to enable a schedule and have a regular backup taken of your site. Click the Scheduled Backup option and set the frequency, start dates, and retention days for the backups. [Figure 23.29](#) shows an automated backup schedule.

Backup options

Scheduled backup ON OFF

Frequency (Days)

Begin 

Retention (Days) 

Always keep at least one export file

Figure 23.29

The Azure Geo-Redundant storage is relatively inexpensive, so it's worthwhile to keep a reasonable retention period for backups. Being able to go back a week or two can be invaluable when a problem occurs.

Once this process is complete, not only should you have a backup of your newly installed site, but you should also have a schedule in place to perform regular backups.

Upgrading to a New DNN Version

It's important to keep up with new DNN releases as they appear. This is not only to gain access to the latest new features but also to ensure that any discovered security vulnerabilities are patched before malicious users start developing exploits. The process for upgrading DNN has changed little from the early versions and wasn't necessarily designed with a product like Azure Websites in mind. Happily, the upgrading process is relatively simple and robust and works well in Azure Websites. Azure Websites does require some changes to the procedure many experienced DNN administrators are familiar with.

Preliminary Upgrade Steps

There are several preliminary steps you need to take prior to upgrading.

Preliminary Step 1: Taking a Backup of Your Installation

The first thing to do in any upgrade situation is to ensure you have a backup of your site in the pre-upgrade state. The previous section shows the detailed steps to create a backup. To be thorough, you might want to test your ability to restore a site from that backup to ensure that your backups are restorable.

Preliminary Step 2: Checking Your Host User Credentials

When you run an upgrade, you'll be asked for your host user and password. It's easy to forget these, and by the time you've started the upgrade process, there's no easy way of determining them. If you kept your credentials text file from installation, you may have them stored.

The best strategy is to take a minute to log onto your DNN installation with the host user and check that you have the right credentials to run the upgrade before you start making any changes.

Preliminary Step 3: Obtaining the Latest Upgrade Package

There are multiple ways to obtain the latest DNN upgrade package. You could follow a link in a community newsletter or follow the link from the Update icon that appears in your site when a new version is released. [Figure 23.30](#) shows when a new version of DNN is released and the update icon appears.

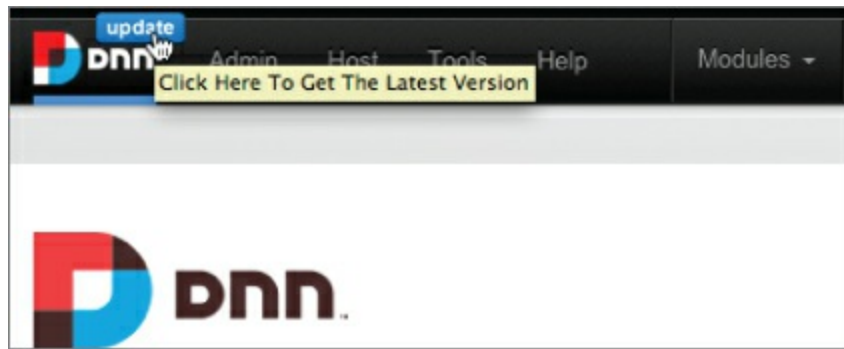


Figure 23.30

Alternatively, you can visit the DNN downloads page at <http://www.dnnsoftware.com/community/download>.

The way to obtain the upgrade isn't as important as making sure you get the correct file. You must obtain the Upgrade package, which has a name like `DNN_Platform_07.03.04_Upgrade.zip`.

The `07.03.04` numbers refer to the new DNN version. Each new version will have a different version number, but the format will stay the same. Download this file to your local computer and unzip it to ensure that no corruption or partial download has occurred.

Preliminary Step 4: Checking Your Dependencies

If you have important third-party extensions in your DNN installation, there may be important updates for the new version you're installing. Take the time to check out the product/project pages for the list of installed extensions and see if there are any notes about required minimum versions or upgrade-specific releases. In some cases, you may need to upgrade an extension, either prior to an upgrade or immediately after.

Preliminary Step 5: Considering Using a Test Copy

Best practice for critical production systems is to take a copy of the production site and restore it to a local copy or even to another Azure website or deployment slot (see “Other Useful Azure Websites Features”). Once a test copy is running normally, run the upgrade process on it and make sure that the upgrade process works correctly and that the site works as normal after the upgrade is complete. By running the upgrade on a test copy first, you can identify any possible problems or compatibility issues. This allows you to complete the production upgrade with high confidence that everything will proceed normally or otherwise make changes to the production environment

in preparation for running the upgrade process.

Upgrading DNN

Once the upgrade preparation is complete, it's time to perform the upgrade. The following steps detail the upgrade process from the point where the target DNN install is ready and the latest upgrade package is downloaded.

Step 1: Uploading the Upgrade Package

The DNN upgrade process consists of copying the contents of the upgrade package over the top of the existing installation files and then requesting the home page of a site in the installation. The Upgrade Wizard is automatically launched when the site detects a mismatch between the file version and the database version. Typically, a site owner or administrator will log onto the server and use an unzip utility to extract the upgrade files. This is not possible with Azure Websites because there is no concept of “logging onto the server.”

To upload the upgrade, it's best to use FTP, as previously explained. I have seen some people unzip the package locally and send the files one at a time over FTP. However, this is a very slow process as there are a very large number of files in a DNN upgrade package. Thankfully there is a built-in utility within the Azure Websites toolset to perform the unpacking of the upgrade zip file. There is a catch, though—many of the files will be locked by the operating system and cannot be overwritten with the unzip utility. You can't stop the website to unlock them because stopping the site also stops the unzip utility.

You can work around this problem by creating a new folder in your site root called `Upgrade`. Typically I'll name the folder using the version number I'm upgrading to. [Figure 23.31](#) shows the created folder with the upgrade package already uploaded.

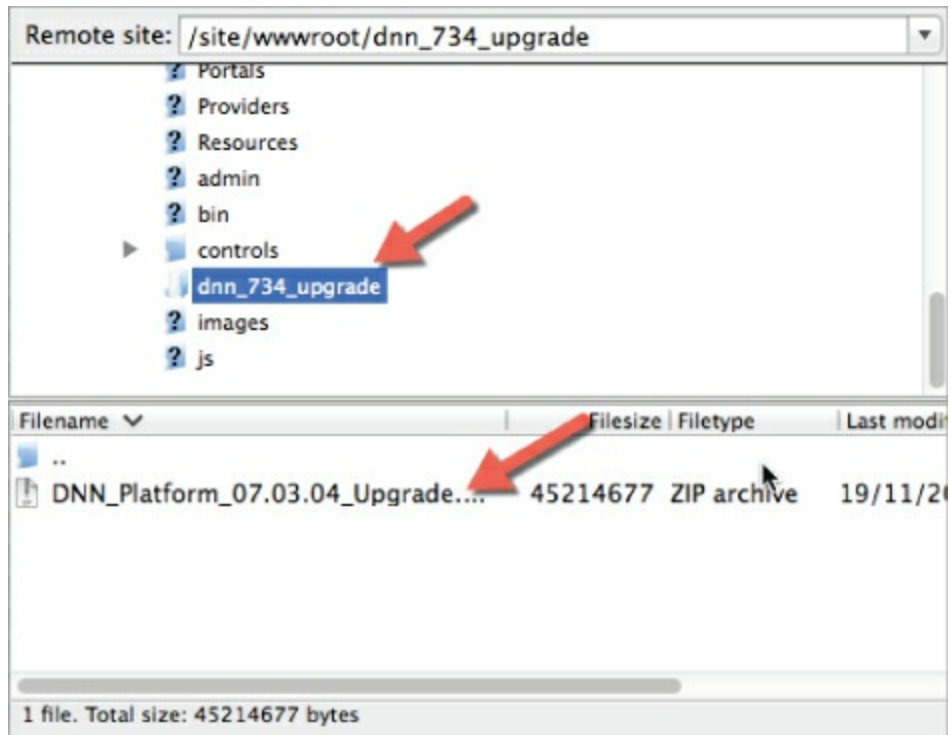


Figure 23.31

At this point the site is still running normally—all that has happened is that a new ZIP file is in a new directory.

Step 2: Extracting and Copying the Upgrade Files

Now, switch back to the Azure site and load your Website blade. Find the Console item and click it. This will open the online console blade, in the root path of your website. The console is a cut-down version of a normal command prompt. [Figure 23.32](#) shows the Azure Console blade open in the Azure site.

The screenshot shows the Azure portal interface for a DNN7DEMO application. The top navigation bar includes 'dn7demo WEBSITE' and 'Console DNN7DEMO'. Below the navigation bar are several control buttons: Add, Browse, Start, Stop, Swap, Restart, Delete, and Reset publish profile. The main content area is divided into several sections:

- Usage:**
 - File System Storage:** DNN7DEMO-PLAN, 0.39%
 - Quotas:** DNN7DEMO-PLAN, Memory Percentage 68%, CPU Percentage 1%
 - Scale:** DNN7DEMO-PLAN, Autoscale Off, Instances 1, Sites 1
 - Estimated spend:** DNN7DEMO-PLAN, Billing data will arrive as you accrue costs
 - Pricing tier:** DNN7DEMO-PLAN, 1 Small Instances
 - STANDARD FEATURES:** Includes icons for storage, security, networking, and monitoring.
- Operations:**
 - Events in the past week:** DNN7DEMO, bar chart showing 25 events.
 - Alert rules:** DNN7DEMO, 2 rules (indicated by a green checkmark).
 - Streaming logs:** A red arrow points to this button.
 - CONSOLE:** A red arrow points to this button.
 - PROCESSES:** A button to view application processes.
 - Backups:** Not Scheduled (indicated by a warning icon).
 - WebJobs:** 0 (indicated by a plus icon).

The console window on the right shows the following output:

```

D:\home\site\wwwroot
> dir
D:\home\site\wwwroot
Volume in drive D is Windows
Volume Serial Number is 746D-E38B

Directory of D:\home\site\wwwroot

11/19/2014 12:10 PM <DIR>      .
11/19/2014 12:10 PM <DIR>      ..
08/11/2014 10:06 AM           22,891 403-3.gif
11/12/2014 12:06 PM           1,298 51Degrees.mobi.config
11/12/2014 11:36 AM <DIR>      admin
11/12/2014 11:36 AM <DIR>      App_Browsers
11/19/2014 11:50 AM <DIR>      App_Data
11/12/2014 11:36 AM <DIR>      App_GlobalResources
11/12/2014 12:06 PM <DIR>      bin
11/12/2014 11:36 AM <DIR>      Components
11/12/2014 12:07 PM <DIR>      Config
11/12/2014 11:36 AM <DIR>      controls
08/11/2014 10:06 AM           2,993 Default.aspx
08/11/2014 10:06 AM          36,603 Default.aspx.cs
11/12/2014 12:06 PM <DIR>      DesktopModules
08/11/2014 10:06 AM           5,430 DNN.ico
11/19/2014 12:11 PM <DIR>      dnn_734_upgrade
11/12/2014 11:36 AM <DIR>      Documentation
08/11/2014 10:06 AM           2,884 DotNetNuke.config
08/11/2014 10:06 AM           832 DotNetNuke.log4net.config
08/11/2014 10:06 AM           1,552 ErrorPage.aspx
08/11/2014 10:06 AM           8,481 ErrorPage.aspx.cs
08/11/2014 10:06 AM           5,420 favicon.ico
  
```

Figure 23.32

Type in `cd {your update directory}` to switch to the directory where you uploaded the upgrade package. Use a `dir` command to confirm the upgrade file is in the folder. Then, use the `unzip` command to unzip the upgrade package. [Figure 23.33](#) shows the use of the `unzip` command.

```
> cd dnn_734_upgrade
D:\home\site\wwwroot\dnn_734_upgrade

> dir
D:\home\site\wwwroot\dnn_734_upgrade
Volume in drive D is Windows
Volume Serial Number is 746D-E38B

Directory of D:\home\site\wwwroot\dnn_734_upgrade

11/19/2014  12:11 PM    <DIR>          .
11/19/2014  12:11 PM    <DIR>          ..
11/19/2014  12:20 PM             45,214,677 DNN_Platform_07.03.04_Upgrade.zip
                1 File(s)      45,214,677 bytes
                2 Dir(s)   53,479,051,264 bytes free

> unzip dnn_platform_07.03.04_Upgrade.zip
```

Figure 23.33

This will take a few moments as the console runs the unzip command and extracts all the files from the upgrade package into the directory. When it is finished, confirm all the upgrade files are now in the folder by using the `dir` command to list them.

At this point, your site is still running as normal, and the upgrade package is not yet applied. The next action will copy the files from the upgrade folder into the site root, putting your site into an upgrade-pending state.

To do this, use `xcopy`, which is available in the console. The command to copy all files and subfolders to the parent directory, without confirming each copy, is shown in [Figure 23.34](#) and listed here:

```
> xcopy *.* ..\ /Y /E
```

```
11/20/2014  12:04 PM    <DIR>          Licenses
11/20/2014  12:04 PM    <DIR>          Portals
11/20/2014  12:04 PM    <DIR>          Providers
11/20/2014  12:05 PM    <DIR>          Resources
                11 File(s)   45,433,575 bytes
                20 Dir(s)  53,399,695,360 bytes free

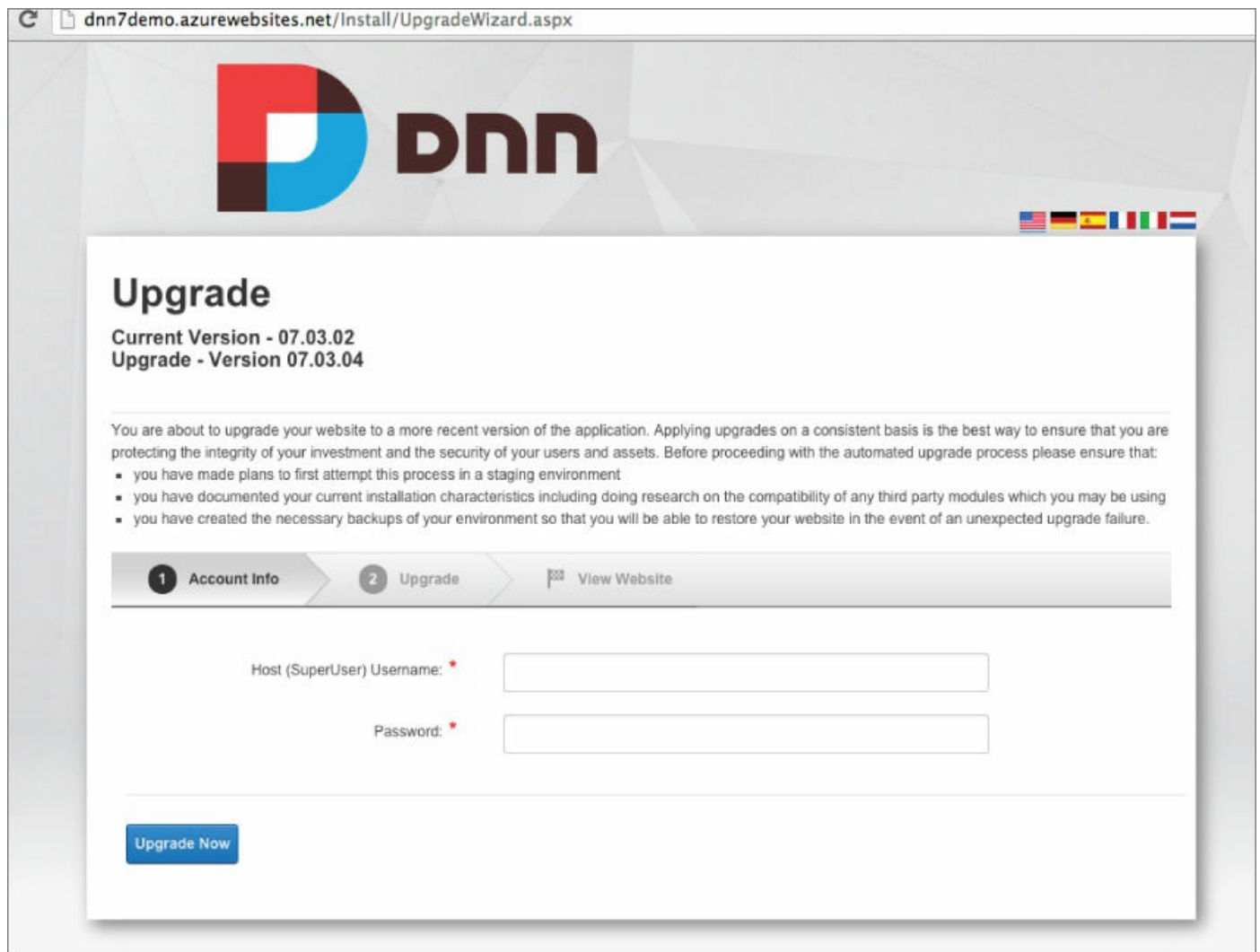
> xcopy *.* ..\ /Y /E
```

Figure 23.34

Once this finishes, your upgrade files are in place on the root path, and you're ready to run the upgrade.

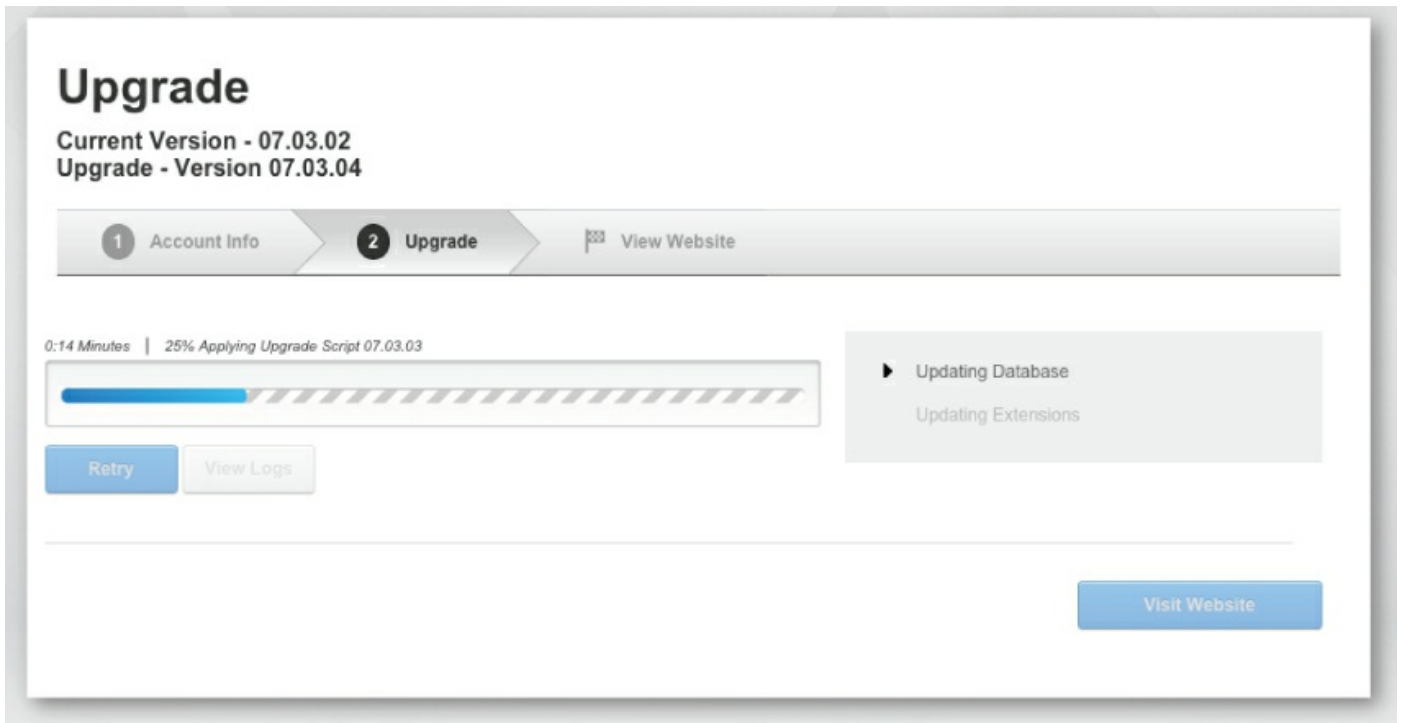
Step 3: Running the Upgrade Wizard

Request the home page of your website, and you'll see the upgrade wizard, as shown in [Figure 23.35](#).



[Figure 23.35](#)

Enter your host username and password and then click the Upgrade Now button, which authenticates you and begins the upgrade process. [Figure 23.36](#) shows the upgrade process in progress.



[Figure 23.36](#)

When the upgrade process completes, click View Website to visit your newly upgraded website. Note that the entire site will have to be restarted and recompiled, so the first request to the site can be sluggish until the new version completely loads. After upgrading, you should no longer see the Update icon on the control bar.

Step 4: Cleaning Up the Upgrade Folder

Now that the site has been upgraded, there is no reason to keep the upgrade folder with the original copies of the upgrade files. Return to the console and navigate to the site root (should show `d:\home\site\wwwroot`). [Figure 23.37](#) shows the process for deleting the temporary upgrade folder.


```
11/03/2014 04:01 PM          221 KeepAlive.aspx
11/03/2014 04:01 PM          1,881 KeepAlive.aspx.cs
11/20/2014 12:04 PM <DIR>      Licenses
11/20/2014 12:04 PM <DIR>      Portals
11/20/2014 12:04 PM <DIR>      Providers
11/20/2014 12:05 PM <DIR>      Resources
          11 File(s)      45,433,575 bytes
          20 Dir(s)      745,213,952 bytes free

> del dnn_734_upgrade /q
D:\home\site\wwwroot

> |
```

Figure 23.37

The console has a strange quirk in that it doesn't allow you to provide additional parameters, so if you try to delete the folder using a normal `del` command, it will do nothing. The reason is that the delete command wants a confirmation when you're deleting an entire folder and subfolders. The trick with this is to add the switch to skip confirmation, as shown in [Figure 23.37](#). The command used is

```
del dnn_734_upgrade /Q
```

The `/Q` switch confirms the deletion and removes the folder and the contents. You should also find the Upgrade Zip package that you uploaded and delete it as well. Good practice dictates keeping only the files you need in your website installation.

Step 5: Performing Testing and Backup

At this point, your site is upgraded and ready for use again. It's important to go check the major features of the site and to test any third-party modules to ensure that the upgrade has not affected them.

At that point, if everything is working as expected, complete your upgrade process by taking another backup of the site.

Moving an Existing DNN Site to Azure Websites

Perhaps you have an existing DNN installation that you want to move to Azure Websites. Although it's difficult to write a detailed step-by-step guide to this process that would match every site, the following steps provide guidelines to work through. Individual problems and issues may crop up along the way, depending on many factors. There are extensive resources available online to help you find solutions to any individual issues you have.

Preparing Your Installation for Migration

These steps are to prepare your existing DNN site for migration to Azure Websites. The steps are for the purpose of ensuring that the compatibility is correct and to ensure the migration process proceeds painlessly.

Preliminary Step 1: Performing a DNN Install on Azure Websites

The first thing you need to do is create a DNN site in Azure Websites. The easiest way to do this is to follow the steps already laid out in this chapter. It's not too important that the site will be replaced by a different version later; the important outcome is that an Azure website, database server, and database and storage account are created.

If you're comfortable with this process, you can skip choosing DNN from the Marketplace and just use a blank website and SQL database. Ultimately, a new Azure website, SQL Database, and the remote connections are required. If you're new to Azure Websites, following the procedure for installing DNN is the easiest way to meet these requirements.

Preliminary Step 2: Upgrading to the Latest DNN Version

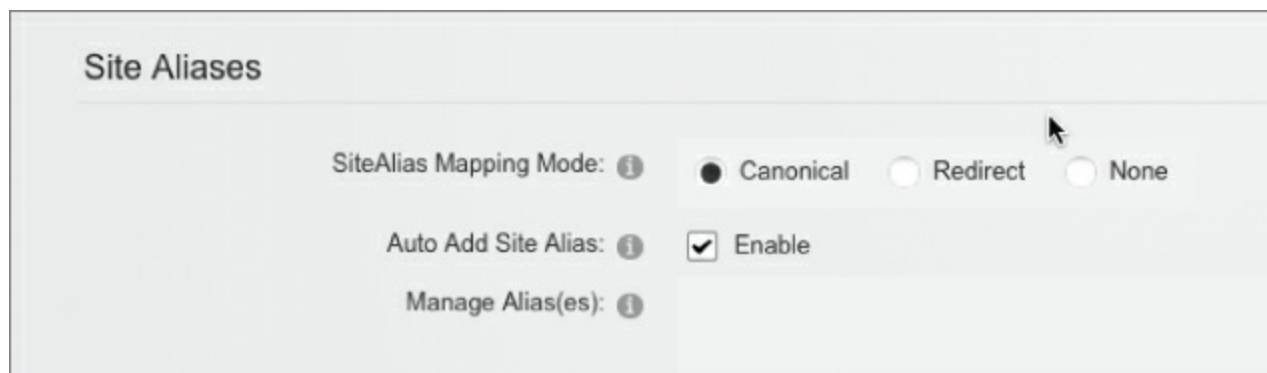
The most important task is to ensure that the DNN installation is on the latest version. This is best practice in general but specifically is required to make sure all the DNN components are Azure compatible. The upgrade process should be run in-place on the current environment.

Do not proceed further with the project until the version is on the latest available DNN package.

Preliminary Step 3: Enabling Auto Add Site Alias

In your existing site, go to the Admin  Site Settings  Advanced Settings and

ensure that the Auto Add Site Alias option is checked, as shown in [Figure 23.38](#).



[Figure 23.38](#)

This will be required when the site becomes available on the new URL in Azure Websites.

Preliminary Step 4: Testing the Database Export Process for Compatibility

Migrating your site to Azure usually means using the Azure SQL database service. It's possible to work around this by using SQL Server on a VM, but this is not a cost-effective solution unless it is a large, high-performance website that demands a stand-alone SQL Server.

While the latest versions of DNN are all compatible with Azure SQL database, it's possible to still have incompatible code, particularly if you have started with an early DNN site and have continually upgraded to the latest versions. This process can leave behind incompatible objects in the system, as the upgrade scripts always have some slight differences with the install scripts.

The easiest way to check for Azure SQL database compatibility is to perform an export/import and see if it works. This is done by accessing your existing database in SQL Server management studio and selecting Tasks ➤ Export Data-tier Application. [Figure 23.39](#) shows the options used to select the export.

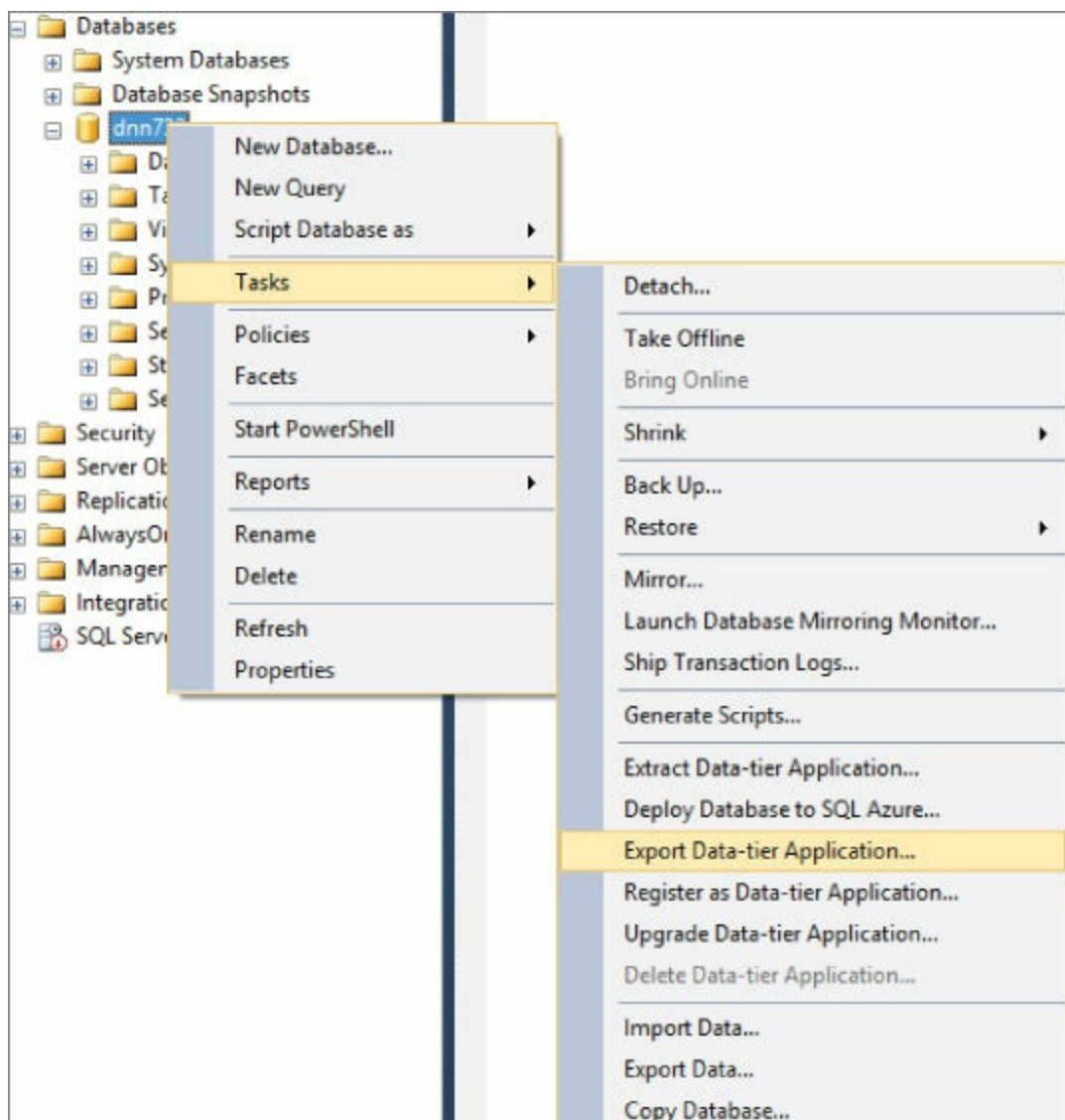
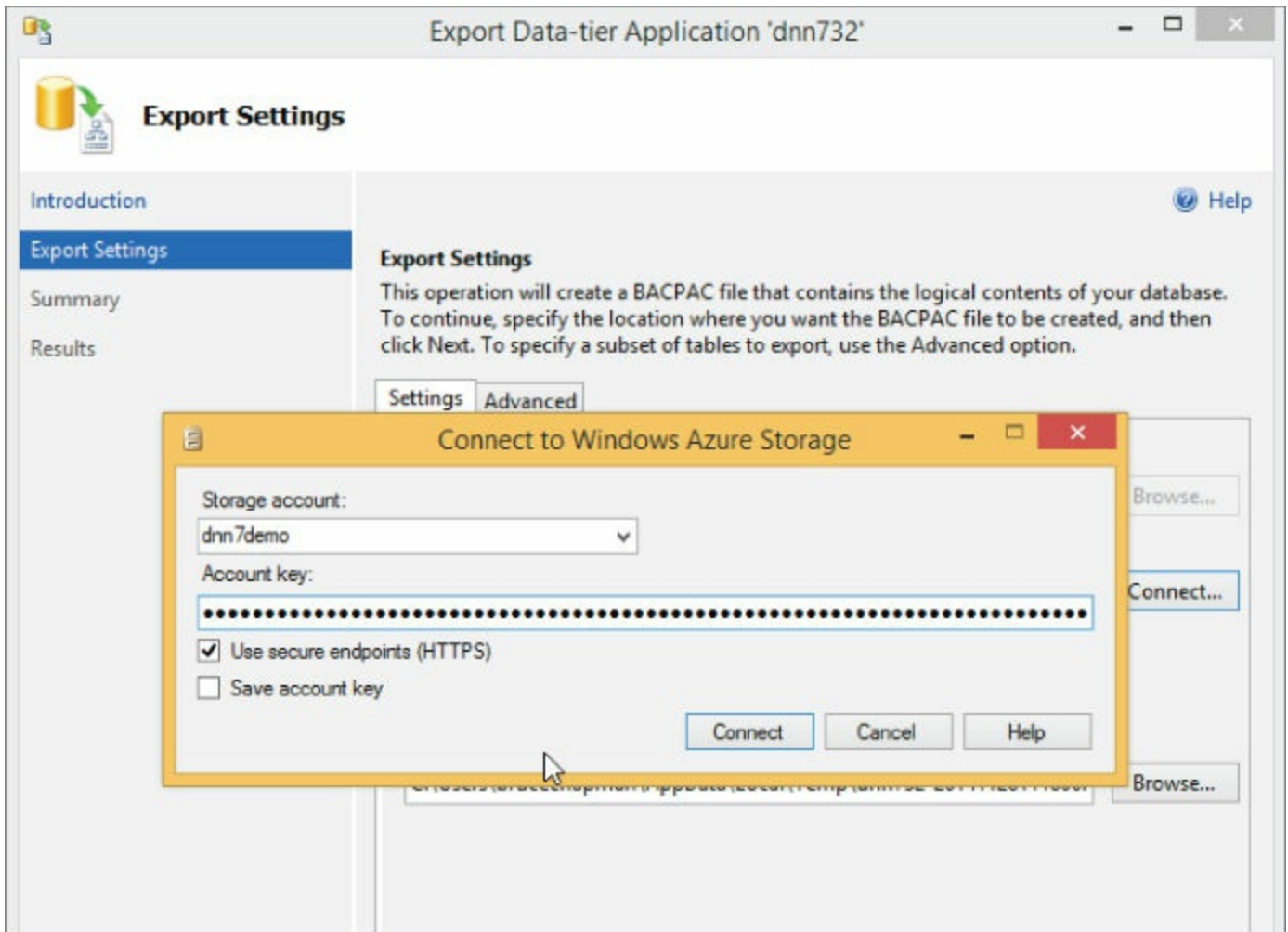


Figure 23.39

This process creates a `.bacpac` file, which is an export of all the SQL Scripts required to re-create the database, plus all of the data required to fill the tables. It isn't a traditional `.bak` backup as you may be used to. These types of SQL Server backup cannot be used to restore to Azure SQL databases.

The easiest way to accomplish this task is to export the `.bacpac` file directly to an Azure Storage account. This option is available in the SQL Server Management Tools database export process. [Figure 23.40](#) shows the entry of the storage account name and key to enable the export of the `.bacpac` file.



[Figure 23.40](#)

This process requires the storage account name and access key. Once this is entered, choose a suitable container in the storage account and complete the process by clicking Next.

It is possible that this process will highlight incompatibilities in the source database and the export will not be able to proceed.

Preliminary Step 5: Testing the Database Import Process for Compatibility

When the database has been exported into a `.bacpac` file, it's time to import it into an Azure SQL database. This can be done in SQL Server Management Studio.

First, connect to your Azure SQL Server with SQL Server Management Studio. You'll need to ensure that you have the server name and administrator credentials. You may also need to allow your IP address through the Azure

SQL Database firewall, which can be done in the Azure SQL Server blade. [Figure 23.41](#) shows how to click the Databases part of the server tree and choose Import Data Tier Application to start the process of importing an existing .bacpac file.

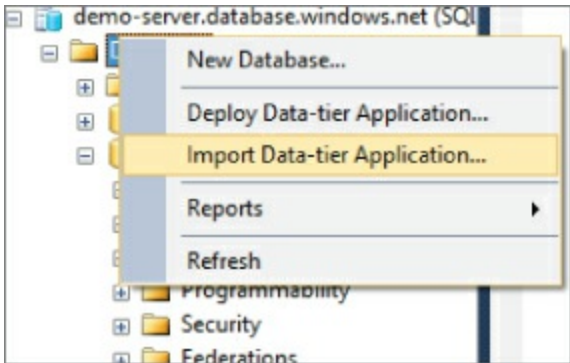


Figure 23.41

Select the option called Import from Windows Azure and select the same storage account, container, and .bacpac file that were created in the previous step. [Figure 23.42](#) shows the example storage account and key being entered to specify the .bacpac import location.

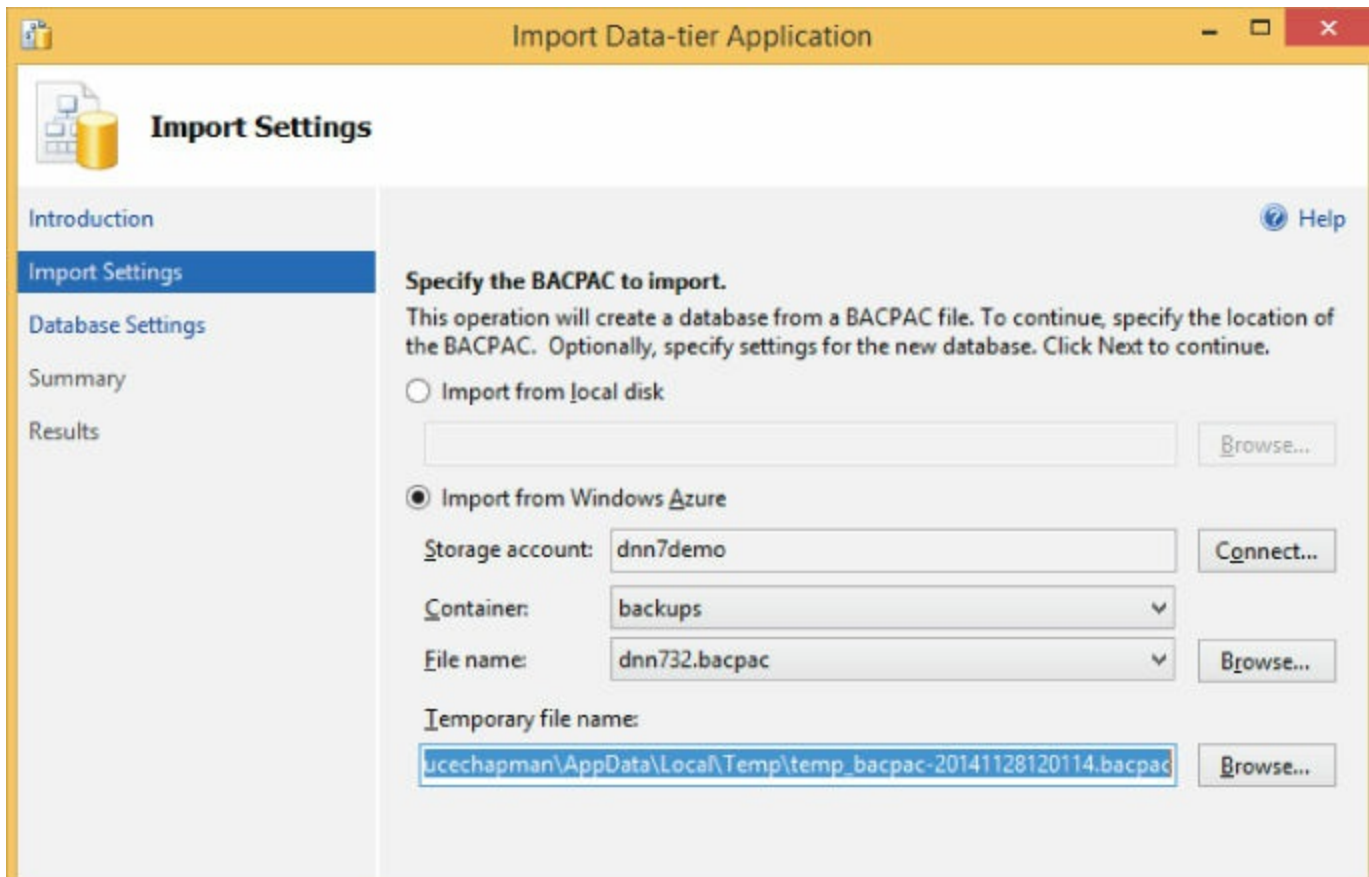


Figure 23.42

Provide a new database name for your import and start the process. If there are compatibility problems, it is at this point that you are most likely to experience them. If the import fails, you will have to repeat steps 4 and 5, fixing the issues as they arise, until the database is imported. Most Azure SQL incompatibilities have simple fixes that can be found with an Internet search.

Performing a Site Migration

If you've followed the preliminary steps, the actual migration is quite straightforward from this point onward. As you have installed a new DNN installation and have upgraded your existing installation to the latest version, you should have two installations with the same version. You should have worked out the process of importing a database and ensuring that there are no compatibility issues.

Step 1: Taking Your Existing Site Offline

While running this process, you need to take your existing site offline so no changes to the files/database occur while you're completing the migration process. You can do this easily by copying an `app_offline.htm` file into the site root of the existing site. This is just a plain HTML file with whatever message you'd like to display. IIS recognizes this file and returns the page for any request to the existing site.

Step 2: Copying and Compressing the Application Files

This first step is to take a copy of your existing site and compress the entire site from the site's `root` folder into a ZIP folder. It is best to use ordinary ZIP compression over other options so that the Azure Websites unzip utility can unpack the files. Your existing environment will dictate how to achieve this step.

Step 3: Uploading and Unpacking Your Application Files

Connect to your target Azure Websites installation and upload the site files. I recommend using the same strategy as outlined in the earlier section on upgrading DNN (“Step 1: Upload Upgrade Package”), which is to upload the compressed files into a folder, unzip them using the Azure console, and then use `xcopy` to extract them into the site root.

After this step is complete, don't attempt to run the site—it isn't ready yet.

Step 4: Importing Your Database Backup

This step is a repeat of preliminary steps 4 and 5, with the outcome of having the site database imported. You should have already gone through this process—what is important is that the latest copy of the site database is imported. If you've made multiple import attempts, be sure to have identified clearly the database copy that was successful. You can rename databases to suit and then delete any failed attempts.

Step 5: Pointing Your Application at the Imported Database

This step involves changing the connection string in your imported site to use the just-imported Azure SQL database.

The simplest way to do this is to open the `web.config` file in Visual Studio Online (covered in a moment) and make the changes. The connection string for the database can be found in the Properties blade of the specific Azure SQL Database blade. Directly edit `web.config` and make this change.

Remember that the connection string may be entered twice in a DNN `web.config` file (in the `<connectionStrings>` section as well as the `<appSettings>` section). [Figure 23.43](#) shows the Database Properties blade with the location of the database connection strings.

The screenshot shows the Azure portal interface for a DNN7demo_Importtest SQL database. The top navigation bar includes 'ONLINE', 'dnn7demo_importtest SQL DATABASE', and 'Properties DNN7DEMO_IMPORTTEST'. Below the navigation bar are icons for 'Delete', 'Restore', and 'Open in Visual Studio'. The main content area is divided into 'Summary' and 'Monitoring' sections. The 'Summary' section shows a diagram of the database architecture with components like 'dnn7demo SQL database', 'demo-ser... SQL server', 'dnn7demo Website', and 'dnn7demo Storage'. A 'Quick start' button is also visible. The 'Monitoring' section contains a 'Resource Utilization' table with a y-axis ranging from 0% to 100%. On the right side, the 'Properties' panel lists various settings: 'PRICING TIER' (S0 Standard (10 DTUs)), 'STATUS' (Online), 'MAX SIZE' (250GB), 'COLLATION' (SQL_Latin1_General_CP1_CI_AS), 'CREATION DATE' (Fri Nov 28 2014, 11:42:45 AM), 'CONNECTION STRINGS' (with a link to 'Show database connection strings'), and 'SERVER NAME' (demo-server.database.windows.net). A red arrow points to the 'CONNECTION STRINGS' section.

Figure 23.43

Step 6: Trying Your Site

At this point, the site should be ready for browsing on Azure Websites. Check this by entering the URL for the Azure website into a browser. Don't use your site domain, which should still be returning an offline message. If everything is set up correctly, the site will load on the Azure Websites URL. This is because the Auto Add Alias option was checked, and the site should determine the new alias and work as expected.

Step 7: Completing the Switchover

The final steps are the same as the final steps in the DNN installation process. You add the DNS entries using the Azure site so that the domain name for the site is configured to point to the Azure website. You need to coordinate this between the DNS provider and the Azure site so that the site can verify that the domains work at the same time traffic starts arriving. Check the SMTP

settings to ensure that it works as expected. Leave your old site with the `app_offline` file in place so that any DNS stragglers receive the “offline site” message and you don't end up with some visitors arriving at the new site and some visitors arriving at the old.

Managing and Troubleshooting Your Azure Website

This chapter has concentrated on the tasks of installing, backing up, and upgrading your DNN installation on Azure Websites. Once installed, a DNN installation requires maintenance and management. This final section of the chapter explains the tools and strategies for troubleshooting and day-to-day operation.

Using Kudu

Kudu is the name of the online tool that runs alongside Azure Websites. It provides both diagnostic information, plus the ability to extend the management environment. It is a vital member of the Azure Websites toolset.

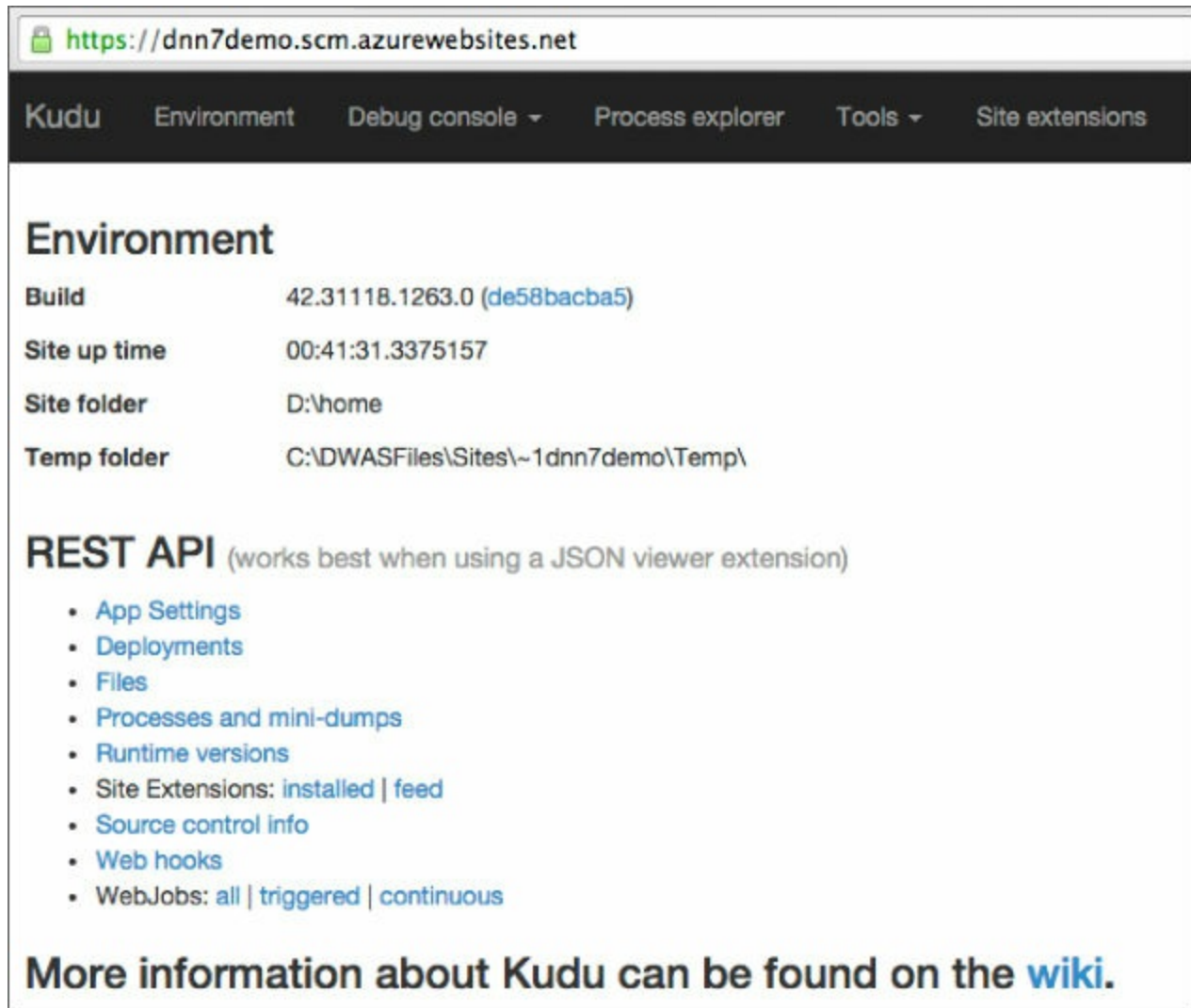
Accessing Kudu is simple; you simply request a variant of your website address by adding `scm` as a subdomain.

Recall that the demo site in this chapter is called `dnn7demo`.

The Azure website is found at <http://dnn7demo.azurewebsites.net/>.

Kudu is found at the address <http://dnn7demo.scm.azurewebsites.net/>.

[Figure 23.44](#) shows the Kudu home page for the example site.



[Figure 23.44](#)

As you can see in [Figure 23.44](#), there are a number of sections within the Kudu services site. The home page provides some basic information about the version and some REST API endpoints you can use if you want to start building automated or remote tools to work with your installation.

Environment

The Environment section lists all the types of information you may need to know when running an application in an IIS environment, particularly if you're writing code to extend the application. Here you'll find items like working directories, current App Settings, CLR versions, and Environment settings.

Debug Console

You are familiar with the Debug Console if you stepped through the section on updating Azure Websites. This provides you with a subset of normal command-line actions that you can perform directly on the file system that contains your DNN installation.

You can also load up a PowerShell command prompt, if you have developed specific PowerShell scripts for your site. Those familiar with PowerShell will start to see the automation possibilities for performing actions within their Azure website.

Process Explorer

The Process Explorer is a very simplified window into the currently running processes within your Azure Websites resources. Generally, when viewing this, you'll see two `w3wp.exe` processes running. That is because one is your actual DNN installation and the other is the Kudu process that you're currently using. These provide good insight into the currently running process and are actually more useful than the standard Windows Server Task Manager, which many people instinctively use to see if their code is running as expected.

[Figure 23.45](#) shows the example `w3wp.exe` process for the example site. This is the actual application process hosting the site in the Azure Websites instance.

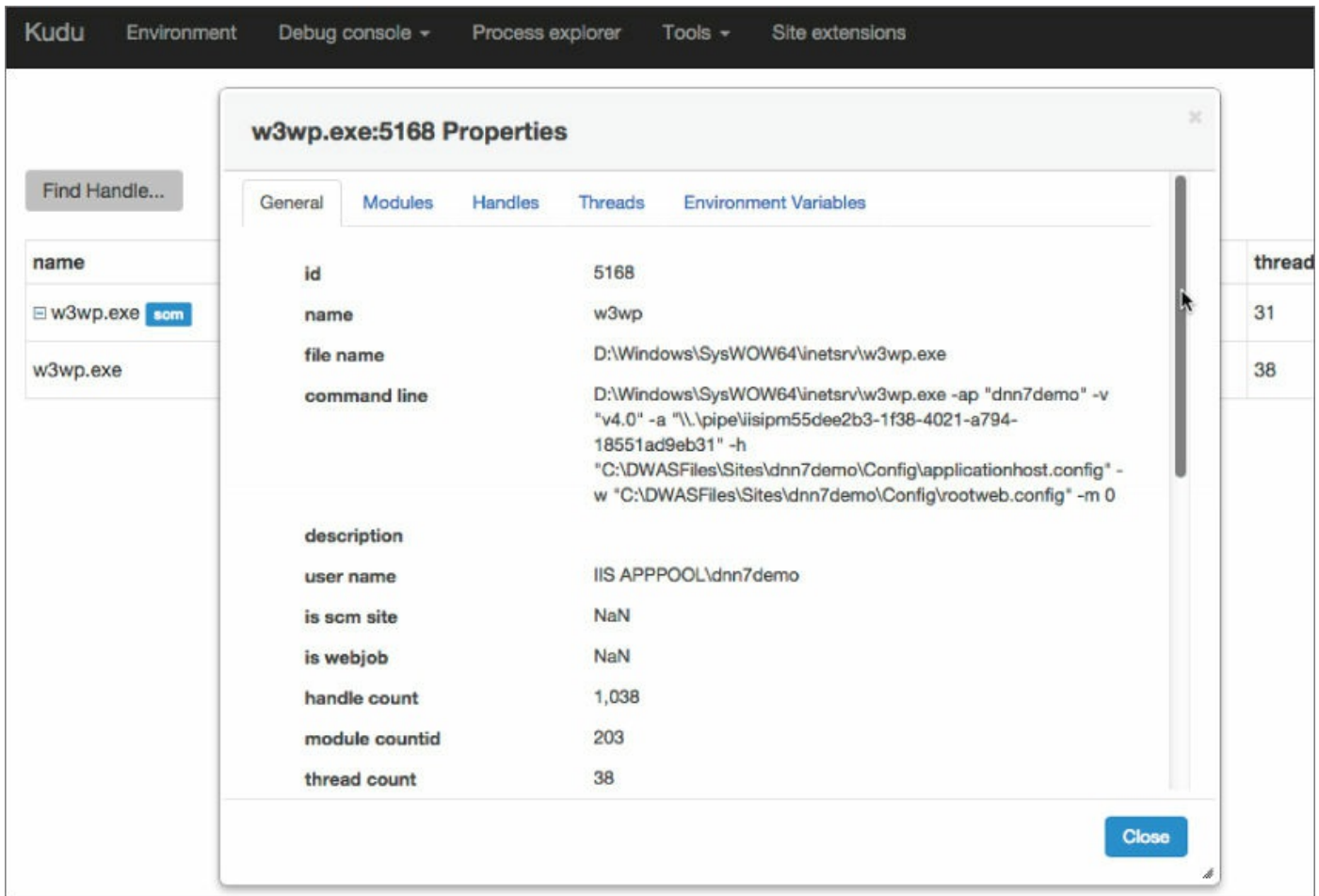


Figure 23.45

The Process Explorer allows you to drill down to the individual thread level. It's not real-time and requires refreshes to view constant updates on how the process is running. It does provide the ability to kill a long-running or unresponsive process. These insights can help understand why an Azure website is behaving in a certain way, by showing the current variables, resource consumption, and loaded assemblies.

Tools

The Tools section is a treasure-trove of useful and interesting tools. The first of these is the diagnostic dump. This incredibly useful utility creates a ZIP archive of current log files and the event log and downloads them instantly to your computer. The Event Log is perhaps one of the most valuable pieces of a Windows Server diagnostic capability, and the diagnostic dump contains the event log information in XML form.

The WebHooks and Deployment script download are of little use to a regular DNN installation. Skipping past them brings you to the Support menu item.

This is another incredibly useful window into your Azure website, which is split between the Observe and Analyze sections.

Support Menu: Observe

The Observe section provides you with near-real time information on requests/second and errors/second. This is useful to observe what the site load currently is and whether there are errors occurring because of that.

Support Menu: Analyze

The Analyze section includes two sub-sections—Diagnostics and Event Viewer.

Diagnostics will read your server web logs and provide insights. This is done by running a diagnosis or by scheduling an analysis to run. When complete, you can directly access the Event Viewer logs, memory dump, and HTTP logs.

Event Viewer is an online way of viewing the process events as you might do if you logged onto Windows Server. If you're tracing errors in the Azure website, this is a good alternative to getting a diagnostic dump because you can simply use the web interface to look through the events being logged.

[Figure 23.46](#) shows the Event Viewer for the Azure website showing application events that have occurred during the execution of the example site.

Azure Websites Support Preview

DIRECTORY Default Directory

SIGNOUT

Select Website **dnn7demo**

Observe | Analyze

Diagnostics **Event Viewer**

Events

Information	11/20/2014 12:46:23 PM	600	PowerShell
Information	11/20/2014 12:46:23 PM	600	PowerShell
Information	11/20/2014 12:46:23 PM	600	PowerShell
Information	11/20/2014 12:46:23 PM	600	PowerShell
Error	11/20/2014 12:27:07 PM	1315	ASP.NET 4.0.30319.0
Error	11/20/2014 12:26:50 PM	1315	ASP.NET 4.0.30319.0
Error	11/20/2014 12:21:52 PM	1315	ASP.NET 4.0.30319.0
Error	11/20/2014 12:20:25 PM	1315	ASP.NET 4.0.30319.0
Warning	11/19/2014 11:50:51 AM	1301	ASP.NET 4.0.30319.0
Error	11/19/2014 11:50:51 AM	1315	ASP.NET 4.0.30319.0

Details

Event code: 4005
 Event message: Forms authentication failed for the request. Reason: The ticket supplied has expired.
 Event time: 11/19/2014 11:50:51 AM
 Event time (UTC): 11/19/2014 11:50:51 AM
 Event ID: 1beee1e79e8545b088eaa082c69e978d
 Event sequence: 2
 Event occurrence: 1
 Event detail code: 50202

Figure 23.46

The last section in Kudu is the Site Extensions. These are tools that you can install into your Kudu environment and enable different functionality. The list is extensive and growing all the time, but notable inclusions include Visual Studio Online, the Azure Websites Logs Browser, and New Relic, which is a very powerful performance analysis tool. [Figure 23.47](#) shows a selection of the Site Extensions available.

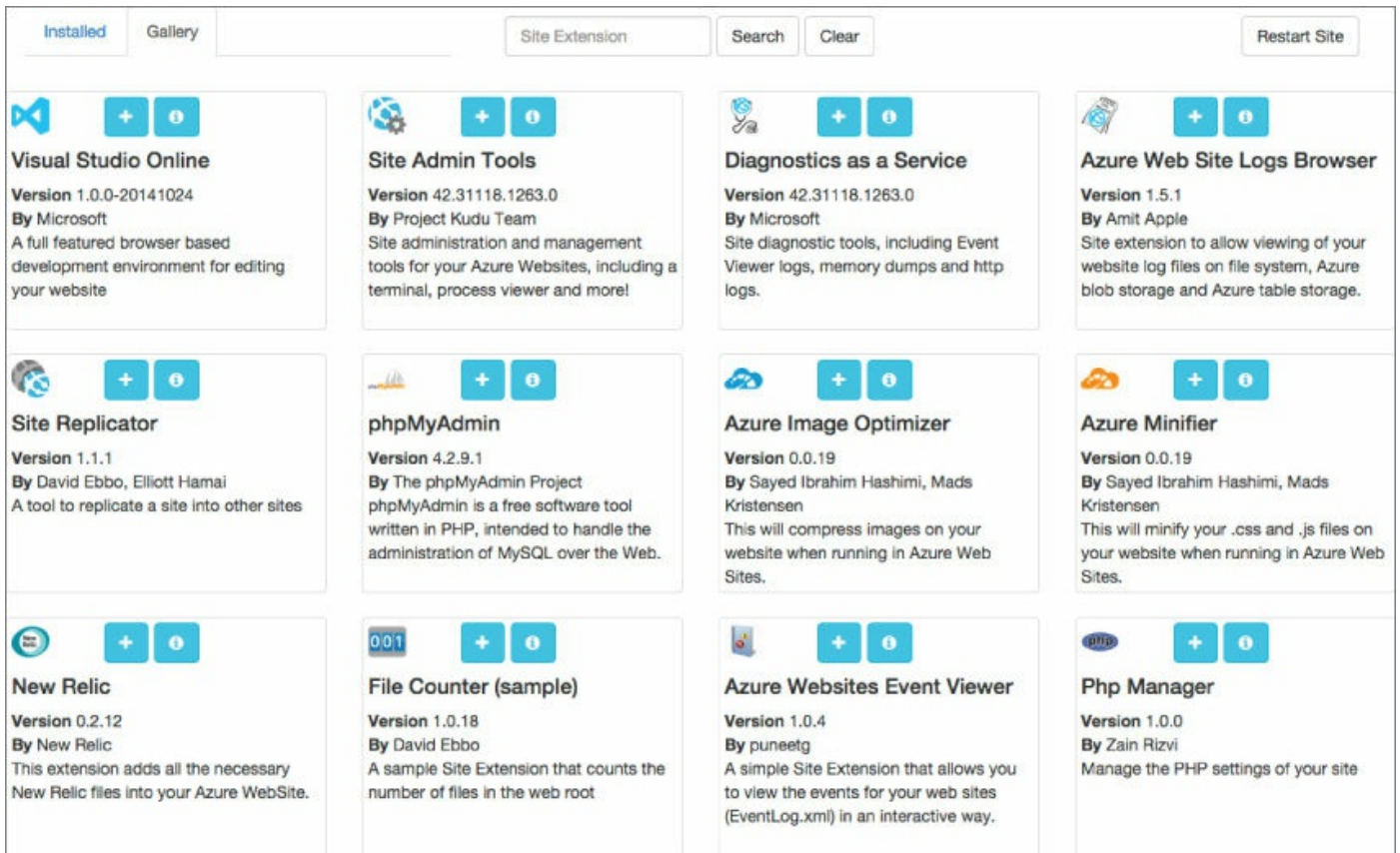


Figure 23.47

Site Extensions can be installed directly by clicking the + icon. Once installed, they can be accessed from the Installed section of the page and run from there. It's important to be careful about installing endless amounts of tools—each one is a set of code that runs and consumes resources in your basic Azure Websites resource plan. In other words, they're sharing resources with the DNN installation, so keep that in mind.

Visual Studio Online

One of the most interesting and powerful extensions in the Azure Websites site extensions gallery is called Visual Studio Online. Not to be confused with connecting your desktop Visual Studio installation to the cloud, this is a full online code editor hosted in the browser.

After installing it from the Gallery, you can view it. The URL is the same as for Kudu with `/dev` on the end. [Figure 23.48](#) shows Visual Studio online with the `web.config` file loaded.

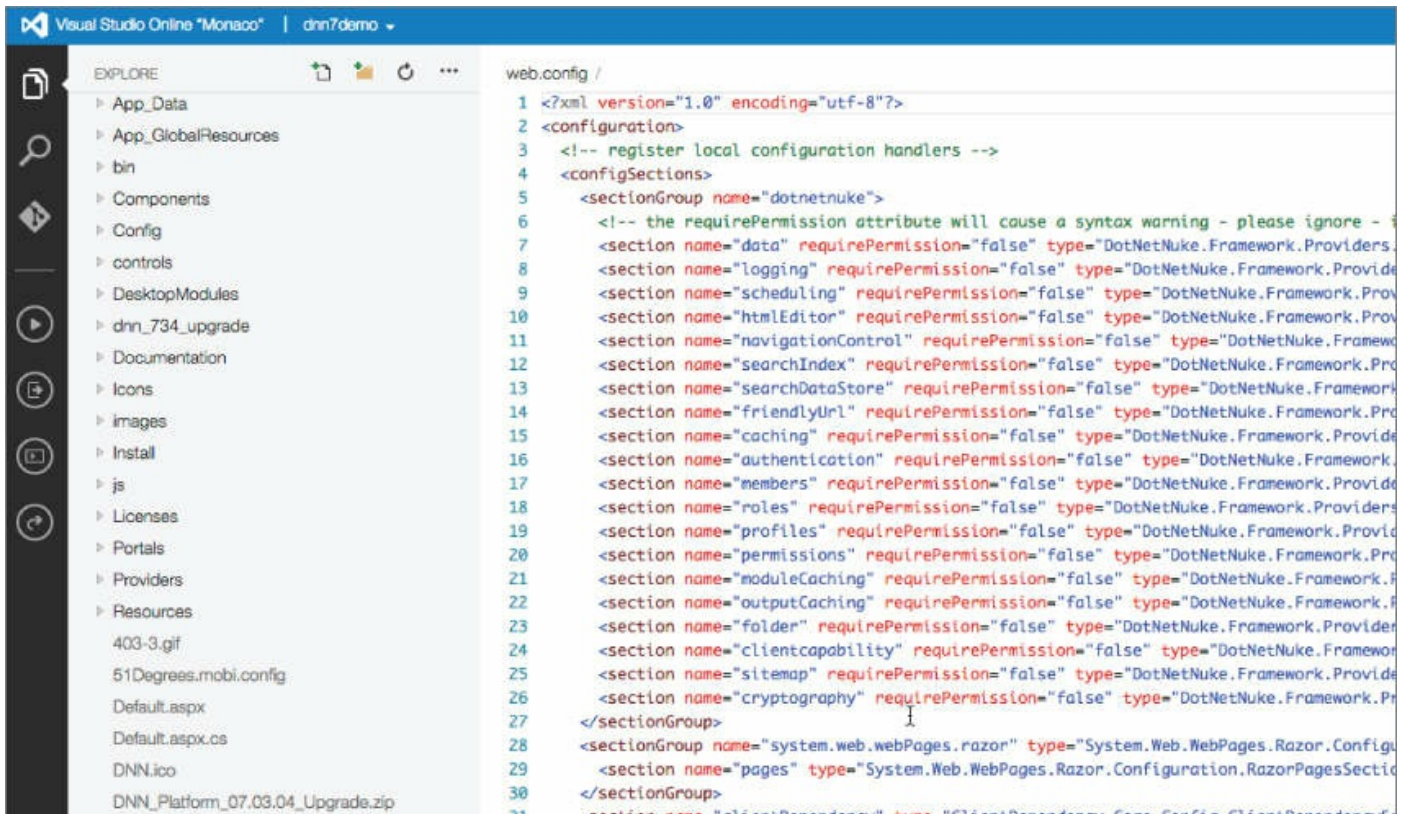


Figure 23.48

As you can see in [Figure 23.48](#), you have a listing of all the files in the site down the left side of the screen. When you click a file, it loads on the right side of the screen. The other options include a command window (similar to the command window accessed from the Azure Site), search, and GitHub integration.

From here, you can edit any of the files in your Azure website installation. Be warned that any changes made are saved automatically and are applied immediately.

This is an excellent resource for making on-the-fly changes to a site. It's immensely useful when you're trying to fix an error. It is best characterized as a “sharp” tool—invaluable for performing certain work but dangerous in unskilled hands.

Other Useful Azure Websites Features

A single chapter is insufficient to explore in-depth all of the available features of Azure Websites. There are many more capabilities to explore. The following is a list of topics that the owner of a DNN install on Azure Websites may want to investigate further.

Autoscaling

In the Standard pricing tiers, websites can be configured to autoscale. Autoscaling is the process of expanding the number of instances available to handle load within a site. This can work either by detecting the CPU load of the existing instance(s) or by using a repeating calendar schedule. All traffic arrives to the Azure website via the Azure Load balancer, and no extra configuration is required to distribute the traffic between instances.

The use of Autoscaling in DNN means that, periodically, the website will run in an effective web farm mode when there are two or more instances serving the site. This may have implications for caching and scheduled task server affinity. It is important to investigate these areas to understand the possible impact. DNN on Azure Websites has been tested with up to 10 instances working together, which is enough power for a standard site to process five million requests per day.

Deployment Slots

The first part of the chapter compared Azure Websites to “IIS as a service.” As any administrator knows, any installation of IIS can support multiple websites. The same is true of Azure Websites, and the concept of deployment slots is how this is surfaced. It's possible through the use of deployment slots to create a copy of their existing site, for either staging or testing purposes. It's possible to stage changes to a deployment slot and then swap the active slot over from one installation of a site to another, giving an instant way of releasing new features.

This works for DNN, as it does for any other site, but as changes in DNN tend to be a single unit comprising database and code changes, it's generally not easy or desirable to modify the database being shared by two instances. For this reason, using deployment slots to update is not suitable, unless a copy of the database is also taken. For sites with relatively read-only content, this is less of a problem. Deployment slots are also an excellent way to create test-only environments by leveraging the resources already purchased for the “main” site.

WebJobs

One core strength of DNN is its ability to create a scheduled task, which is a set of work that runs independently of a user action. Many DNN installations

have separate scheduled tasks to run. There is another alternative in Azure—the concept of WebJobs. A WebJob is a unit of code that can be tasked to run in the background—on demand, continuously, or on a set schedule. They are more flexible and easier to create and configure than DNN scheduled tasks. In addition, WebJobs can be run using popular scripts such as DOS `batch/cmd` files, PowerShell scripts, executables, or even other languages such as Python, PHP, or JavaScript. WebJobs are included free with Azure Websites.

Application Monitoring

It's important to understand how your DNN installation is running. Understanding performance cuts across several levels—from understanding how many requests the application is handling to understanding how the code itself is performing. This type of end-to-end view of application performance is possible with New Relic, which can be switched on for an Azure website with built-in integration to the Azure site.

The basic New Relic account gives 24 hours of history and provides the ability to upgrade to higher levels of account access, with more in-depth analytics and a longer performance history.

In addition to New Relic, Microsoft is entering the monitoring space by expanding the product known as Application Insights. This gives performance data of the Azure website and the ability to set up a `ping` URL (called a webtest) that keeps the DNN site alive and sends an alert when the application stops responding. When setting these up in DNN, it's recommend to use the built-in pinging URL `/keepalive.aspx`.

Other Azure Resources

Creating an Azure website means you can start leveraging the other Azure services. DNN uses the concept of *folder providers*, which allows the location of external files on cloud storage providers. Several vendors market Azure folder providers, which allow data to be stored on the Azure storage account created in this demonstration.

Once data is stored in an Azure storage account, it's a relatively simple transition to start using the Azure CDN to lower bandwidth costs and improve visitor download speed, particularly if you're distributing large amounts of files such as images or documents.

For the purposes of video, there is Azure Media Services to handle storage

and encoding and streaming of video and audio files. These can easily be integrated into a DNN installation if this type of functionality is required.

The list of Azure services grows regularly, and many of the technologies can be applied to solve specific problems that the owner of a DNN site may have. It's definitely a good idea to become familiar with the depth of services when faced with a new requirement.

Summary

In this chapter you learned how to create a new Azure subscription, create an Azure website, and install DNN 7. The chapter covered the recommended deployment model for DNN on Azure. It also explained how to back up, upgrade, and manage your DNN installation in Azure Websites. You learned the basics on migrating an existing DNN website to Azure Websites and where other Azure services might be leveraged when building professional websites with DNN.

Appendix A

Resources

Although DNN is rich in “out-of-the-box” functionality, its real power comes in the ability to extend and enhance it via one of its many extension points. This appendix explores developer resources, popular extensions, and useful resources at www.dnnsoftware.com.

[Table A.1](#) describes tools that are useful for developing extensions.

Table A.1 Developer Tools

Tool	Description
Fiddler— http://fiddler2.com	A free web debugging proxy (Burp is another excellent free alternative). Fiddler allows developers to log all HTTP(S) traffic between the client (DNN) and server. Particularly useful for debugging Service Framework (WebAPI) requests.
dotPeek— www.jetbrains.com/decompiler/	A free .NET compiler. Useful for navigating, searching, and

	decompiling assemblies when source code is not handy.
Christoc's DNN module development template — http://christoctemplate.codeplex.com/	Very useful, free, and popular DNN module development templates.
MVP DAL2 template — http://www.dnnsoftware.com/forge/enterprise-dnn-c-mvp-module-template-for-visual-studio/view/extensiondetail/project/dnmvptemplate	A C# Visual Studio template for DotNetNuke 7 Module Development using the new DAL2 Data Access Layer and implementing the MVP and Repository design patterns.
Beyond Compare — www.scootersoftware.com	Compares files (and file contents) and folders to help identify any changes.
Filezilla — http://sourceforge.net/projects/filezilla/	Powerful, free open source FTP client.
Sourcetree — www.sourcetreeapp.com/	Free Git (and mercurial)

client for Windows. Now that the DNN Platform team's work is all on GitHub, this is a very useful tool if you want to work with the latest DNN code or contribute your own changes. It's also very useful for your own development needs.

Visual Studio tools for GIT

—<http://visualstudiogallery.msdn.microsoft.com/abafc7d6-dcaa-40f4-8a5e-d6724bdb980c>

A team explorer extension that provides source code integration with GIT. Note: these extensions are for VS .NET 2012 only as VS .NET 2013 comes with them preinstalled.

Firebug—<https://getfirebug.com/>

Today, all the major browsers have built-in web development tools (typically started by pressing F12) However, the oldest and still the best (primarily due to its extensibility) is Firebug, which is vital for debugging JavaScript, tweaking markup, and analyzing everything from CSS inheritance to network performance.

Visual Studio Community 2013

—<http://www.visualstudio.com/products/visual-studio-community-vs>

A community version of Microsoft's flagship development tool. With virtually everything that the commercial

	<p>VS .NET product contains, and much more than previous efforts such as WebMatrix, this is an essential download.</p>
<p>MakeDNNSite—http://mikevdm.com/DNN</p>	<p>Free DNN installation helper. Very useful free tool for setting up DNN on your local machine. It automates all the necessary steps, including configuring IIS and SQL, downloading the file, and automating the installation.</p>

[Table A.2](#) describes a number of popular extensions. This table covers only free (and usually open source extensions), and I've chosen to highlight popular, active projects.

NOTE

The DNN store (<http://store.dnnsoftware.com>) has more than 10,000 extensions available for those whose projects have budget to spend.

Table A.2 Popular Extensions

Extension
DNN Blog— www.dnnsoftware.com/forge/view/ExtensionDetail/project/dnnk
DNN Events — www.dnnsoftware.com/forge/view/ExtensionDetail/project/dnnevents
NBStore— www.dnnsoftware.com/forge/view/ExtensionDetail/project/NBSto
Forms & List— www.dnnsoftware.com/forge/view/ExtensionDetail/project/d

2Sexy Content

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/SexyContent

DNN Glimpse

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/dnnGlimpse

DNN Content localization tools

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/DNNCLTools

DNN CookieConsent

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/DnnCookieConsent

40Fingers StyleHelper

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/StyleHelper

CKEditor

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/dnnckeditor

Advanced Control Panel

—<http://oliverhine.com/DotNetNuke/Administration/AdvancedControlPanel>

DNN Azure accelerator

—www.dnnsoftware.com/forge/view/ExtensionDetail/project/dnnazureaccel

[Table A.3](#) lists a number of useful resources provided on

www.dnnsoftware.com. Although many of these will be known to experienced DNN users, some aren't as well known but are extremely useful.

Table A.3 [dnnsoftware.com](http://www.dnnsoftware.com) resources

Resource	Description
Forge— www.dnnsoftware.com/forge	More than 400 free, open source DNN projects.
Forums— www.dnnsoftware.com/forums	Forums on all aspects of DNN.
Wiki— www.dnnsoftware.com/wiki	More than a half-million words of useful community-sourced documentation.
Online Help— www.dnnsoftware.com/Help	An online (searchable) version of the latest DNN documentation.
Community Voice— www.dnnsoftware.com/voice	Share and vote on ideas for DNN. The DNN Platform team uses this to help decide on enhancements for future releases.
Community Exchange — www.dnnsoftware.com/answers	Ask and answer questions on DNN. This module supports voting, so the best responses float to the top and awards points to the answerer.
Video library— www.dnnsoftware.com/videos	Contains hundreds of videos covering a wide array of DNN topics.

Nightly builds

—www.dnnsoftware.com/Platform/Build/Nightly-Builds

This page hosts a copy of a nightly build of the latest in-development version of DNN (particularly useful for those who want to play/test but don't want to have to use VS .NET).

EVS—<http://evs.dotnetnuke.com/>

The Extension Verification Service (EVS) allows developers to easily upload extensions for testing. Once extensions are uploaded, the service unzips the packages, performs the verification checks, and returns a summary. The summary information—including any errors, warnings, and info messages—can then be downloaded, emailed, or shared. SQL Azure testing is also supported.

Manuals

—www.dnnsoftware.com/Community/Download/Manuals

Download manuals from various releases/versions for offline reading/printing.

UX guide—<http://uxguide.dotnetnuke.com/>

Documentation and examples of DNN 6

user experience
“patterns.”

Appendix B

System Message Tokens

System tokens can be used when customizing portal email templates and HTML module content (although they are also used for other purposes). The tables in this appendix identify and briefly describe the properties of each token. Recall that token properties may be referenced in email templates using the pattern `[Token:Property]`. Only properties that are publically accessible are listed.

[Table B.1](#) describes the Host properties that are available for the token `Host` (such as `[Host:HostTitle]`).

Table B.1 Standard HostSettings Properties

Property Name	Description
HostTitle	Name of the hosting account. This name is used throughout the site for identifying the host.

`PortalSettings` site properties are available for the token `Portal` (such as `[Portal:PortalId]`). [Table B.2](#) describes them.

Table B.2 Standard PortalSettings Properties

Property Name	Description
BackgroundFile	Graphic file used for the site background.
DefaultLanguage	Default locale of the website. This determines the language used when anonymous users visit the site.
Description	Website description. This information is included in the meta tags used by search engines.
Email	Email address for the site administrator (this is generally set to a support email address).
FooterText	Information displayed in the copyright skin object.
HomeDirectory	Folder name associated with the current site. The name is a relative path to the site root directory.
KeyWords	Specific meta tag keywords.
LogoFile	Graphic file used for displaying the portal logo.

PortalId	ID of the current portal.
PortalName	Name of the current site. This name is used for branding the site.
SiteLogHistory	How many days to keep the SiteLog history for the site.
URL	Returns the default site alias for the site.

[Table B.3](#) describes the `UserInfo` properties available for the token `User` (such as `[User:UserID]`).

[Table B.3](#) Standard UserInfo Properties

Property Name	Description
DisplayName	User's display name
FirstName	User's first name
FullName	Obsolete, returns <code>DisplayName</code>
LastName	User's last name
PortalID	<code>PortalID</code> to which this user belongs
UserID	Unique identifier for a specific site user
Username	Logon name of the specific user

[Table B.4](#) describes the `UserMembership` properties available for the token `Membership` (such as `[Membership:Password]`).

[Table B.4](#) Standard UserMembership Properties

Property Name	Description
Approved	If the user's account has been approved for access to the website
Email	Email address of the user
CreatedOnDate	Date/time when the user account was created
IsOnline	Provides information whether the user is online
LastActivityDate	Date/time the user was last active
LastLockoutDate	Date/time the user was last locked out
LastLoginDate	Last date/time the user logged in to the site
LastPasswordChangeDate	Date/time the user last changed his or her

LockedOut	password If the user's account has been locked due to potential security issues
UpdatePassword	Should the user change his or her password
Username	Login name of the user

[Table B.5](#) describes the `UserProfile` properties available for the `Profile` token (such as `[Profile:FirstName]`).

[Table B.5](#) Standard UserProfile Properties

Property Name	Description
Biography	User's biography.
Cell	Mobile phone number.
City	City.
Country	Country where the user lives.
Fax	Fax number.
Facebook	Facebook profile URL for the user.
FirstName	User's first name.
IM	Instant messenger contact ID.
LastName	User's last name.
LinkedIn	LinkedIn profile URL for the user.
MiddleName	User's middle name.
Photo	User's profile picture. This returns a relative URL to the image file.
PostalCode	Postal code for the user's mailing address.
Prefix	User's prefix.
Region	State, province, or region for the user.
Skype	Skype handle for the user.
Street	Street address.
Telephone	Telephone number for the user.
Twitter	Twitter handle for the user.

Unit	Apartment, post office box, or suite for the user's address.
Website	Personal or corporate website for the user.

Apart from the properties in Table B.5, all custom profile properties are also accessible through the token system. Display of the token is managed by a staggered permission system. First the site administrator can define whether end users can set profile property visibility themselves. If so, end users can set the visibility of each profile property in the user profile editor.

PROFESSIONAL DNN7 OPEN SOURCE .NET CMS PLATFORM

**Shaun Walker
Bruce Chapman
Cathal Connolly
Peter Donker
Israel Martinez
Charles Nurse
Chris Paterra
Clinton Bland
Nathan Rover
Will Strohl
Erik van Ballegoij
Scott Willhite
Ralph Williams
Mitchel Sellers
Dennis Shiao**



Professional DNN7: Open Source .NET CMS Platform

Published by

John Wiley & Sons, Inc.

10475 Crosspoint Boulevard

Indianapolis, IN 46256

www.wiley.com

Copyright © 2015 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-85084-8

ISBN: 978-1-118-85079-4 (ebk)

ISBN: 978-1-118-85086-2 (ebk)

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2015930543

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with

any product or vendor mentioned in this book.

For the Microsoft open source community.

About the Authors

Shaun Walker has 20+ years professional experience in architecting and implementing large-scale software solutions for private and public organizations. Shaun is the original creator of DNN, a web content management system for ASP.NET that has cultivated the largest and most successful open source community project native to the Microsoft platform. Based on his significant community contributions, Shaun has been recognized as a Microsoft Most Valuable Professional (MVP) since 2004 and an ASPInsider since 2005. He was recognized by Business in Vancouver in 2011 as a leading entrepreneur in their Forty Under 40 business awards.

Bruce Chapman has been an IT professional for 20 years across multiple industries, working with a range of technologies. He has been involved in the DNN community for the past eight years as a contributor, commercial vendor, blogger, and frequent presenter at DNN conferences and community events. Bruce joined DNN Corporation in 2012 after the acquisition of the well-known URL Master software for inclusion within the DNN Platform and works as a product manager with a current focus on distributing DNN on cloud platforms.

Cathal Connolly has worked with DNN since its earliest days. He works as a senior engineer at DNN Corporation, focusing primarily on the Platform project, as well as heading up the Security team. He's been a Microsoft MVP for 10 years, starting out in VB.NET before migrating to C# and finally to ASP.NET.

Peter Donker is the founder and owner of Bring2mind, makers of Document Exchange: a document management module for the DNN platform since 2004. He has been a DNN core team member (now DNN MVP) since 2007 and spends a significant amount of his time writing open source extensions or improving the platform itself. In 2008 he founded the DNN community internationalization team that acts as a group of consultants to the DNN development team concerning the use of DNN outside the English-speaking world. He currently still works on this team and the architecture team that evaluates contributions made to the platform through GitHub. He also is a member of the DNN Steering committee that provides oversight for the DNN platform roadmap and community relations. In addition to his work on these teams, he is a cofounder and the current president of DNN Connect, an association of DNN community members that aims to defend the interests

of the DNN community and acts as a platform for community activities such as organizing DNN events and coordinating the development of specific DNN extensions.

Israel Martinez was part of the Product Management Team at DNN Corporation for almost five years. He was part of the evolution of the product from being a platform to providing business solutions such as Evoq Content 8. He holds a bachelor's of mechanical engineering, a master's of e-business, and a master's of software systems. He has spoken at various DNN conferences, including DNNWorld 2013 and the Southern Fried DNN. Israel has a strong passion for business solutions, efficiency, and research, and he uses these skills on a daily basis.

Charles Nurse is the chief architect at DNN Corporation, responsible for the overall architecture of the DNN Platform and Evoq products. He has been a Microsoft MVP in ASP.NET for eight years and an ASPInsider for seven. He has spoken at many major conferences including DevConnections (www.devconnections.com), DevTeach (www.devteach.com), VSLive (www.vslive.com), Microsoft Tech Days, and DevReach (www.DevReach.com). In addition, he is a frequent speaker at user groups and code camps, including VBUG, .NET BC, VanTUG, SeaDUG, and Vancouver Tech Fest. He has created two video courses on DNN for Pluralsight — *DotNetNuke Fundamentals* and *DotNetNuke Module Development*.

Chris Paterra is the senior product manager of the Evoq line of products at DNN Corporation. Originally a part of the initial DotNetNuke project, Chris has over a decade of experience using and developing DNN products.

Clinton Bland is the sales engineer and training manager at DNN Corporation where he provides technical support and product demonstrations for the sales team as well as both internal and external product training. He is a very active member of the DNN community as he is the vice president and evangelist for the Charlotte, NC, based DNN user group and he frequently blogs on DNNSoftware.com. He has helped organize two DNN Community Conferences (Charlotte Day of DNN and Southern Fried DNN) and regularly speaks at DNN conferences, user groups, and universities. In the fall of 2012 he won both DNN Superfan and DNN MVP awards. He was also an assistant baker for the first-ever DNN cake!

Ashish Prasad is the senior manager of development at DNN Corporation,

primarily responsible for the development process in DNN's Evoq products. He has been developing and architecting software for more than 18 years, with over 6 years creating commercial solutions using DNN as a platform. Ashish frequently blogs at dnnsoftware.com. Some of his direct contribution in DNN includes the areas of social, messaging and notification, gamification, mobile, and most notably search. Ashish has had senior engineering positions in both Fortune 500 companies and startups, with verticals ranging from banking, insurance, manufacturing, digital marketing, healthcare, and retail. Ashish holds a bachelor of technology in chemical engineering from the esteemed IIT, Varanasi, India, and an MBA in technology management from the University of Phoenix. He also holds a CISSP (Certified Information Systems Security Professional) designation. He is on Twitter at [@ashishprasad](https://twitter.com/ashishprasad).

Nathan Rover is an experienced developer with more than 17 years in the Internet industry. He has been working with the DNN Platform for more than 9 years and has been working for DNN Corporation as the principal ecommerce engineer since January 2011. During that time he has been the lead developer for the main corporate site (www.dnnsoftware.com), The Store (store.dnnsoftware.com), and Extension Verification Service (eve.dnnsoftware.com).

Will Strohl is an author and technologist in the San Francisco area, specializing in DNN and the Microsoft.Net stack. Will has held positions ranging from help desk technician to being the director of technology and is a former employee of DNN Corporation, having run the evangelical, sales engineering, and education programs. Today, Will is the director of product management and co-owner at a very exciting e-commerce software company, Hotcakes Commerce (www.HotcakesCommerce.com). He is also the co-host of the monthly DNN Hangout video podcast. In his spare time, Will maintains nearly 30 open source projects, speaks at various user groups and code camps, mentors others in leadership and personal branding, plays Call of Duty, and enjoys outdoor activities and running.

Erik van Ballegoij is a senior .NET consultant and DNN expert. He is currently working for Arrow Consulting and Design, a gold-level DNN partner. He is also a DNN MVP and is very active in the DNN community. Until September 2014 he was senior developer at DNN Corporation, based in the European office in Amsterdam, The Netherlands. As a member of the engineering team, his activities focused on localization, ranging from feature

design and implementation to coordination of translations. Apart from that, he was also the main resource for the DNN Developer Support program, a service available to customers and partners. Erik is a longtime member of the DNN community, drawn to it in the late days of DNN 2. Localization has always been an area of interest, resulting in the translation of DNN 3 and 4 in Dutch and the development of various popular localization products. He joined the core team in 2007 and was also awarded Microsoft MVP in that same year, the first of a total of five. In 2011 Erik joined DNN Corporation as a sales engineer/support representative. The customer interaction experience has been invaluable in his role as senior developer. Erik is board member of the Dutch DNN user group and founding member of the European DNN professionals network.

Scott Willhite is cofounder of the DotNetNuke open source project and DNN Corporation. A technologist by trade, he studied computer science and information systems management at Baylor University and honed those skills with Accenture into enterprise architecture expertise for the likes of USAA, Bank of America, and American Express and in fields ranging from healthcare to consumer products. He has co-authored three previous books including *DotNetNuke for Dummies*, has been recognized with Microsoft's MVP award on multiple occasions, and has spoken at several major and minor conferences including DevConnections, DNNWorld, and various DNN user group events. Scott is an avid supporter of DNN's "abundance mentality" with more than a decade of experience in community development, volunteer coordination, and relationship counseling. He is most gratified when his love of people and technology collide to improve the quality of life.

Ralph Williams, Jr., is the web developer manager at Arrow Consulting and Design as well as a user experience and graphic designer. He leads a team of front-end developers who specialize in building large DNN websites. He has been involved in the DNN community for nearly 10 years, speaking at several DNN conferences including DNN World, code camps, and DNNCon (formerly Day of DotNetNuke). He combines his front-end development and ux design knowledge to create usable experiences for enterprise-level web applications.

Mitchel Sellers is CEO/director of development at IowaComputerGurus, Inc., responsible for day-to-day business operations, software architecture, and training needs. He has been a Microsoft MVP in C# for six years and DNN MVP since the inception of the program. He has given more than 150

talks at user groups, code camps, and conferences large and small across the world. He previously authored *Professional DotNetNuke Module Programming*, co-authored *Visual Studio 2010 Six-In-One*, and has served as a technical editor on numerous other titles. For more information on Mitchel and his hobbies, visit his website <http://www.mitchellsellers.com>.

Dennis Shiao is the director of content marketing at DNN Corporation. Responsible for a program that generates leads and drives revenue, Dennis himself is a content machine, personally creating or overseeing all corporate blog posts, white papers, data sheets, and presentations available on the DNN website and SlideShare. A prolific writer, he is also the author of the book *Generate Sales Leads with Virtual Events* and is a contributing author to the book *42 Rules of Product Marketing*.

About the Technical Editors

Sebastian Leupold earned a degree in business engineering from Karlsruhe University and in 1997 founded the corporation gamma concept - Gesellschaft für Aktuelle Management-Anwendungen mbH. The company focuses on software solutions for PCs and the Internet, especially on web applications based on the DNN open source platform. gamma concept took part in several national and international research projects regarding IT applications in SME including multilingual solutions. Since 2007 Sebastian has been a member of the DNN Core Team and co-founder of the German DNN User Group as well as the European Network of DNN Professionals. Besides supporting the community with nearly 50,000 forum posts, he presented at various conferences in Europe and North America. A couple of features and improvements for the DNN Platform were contributed by him. His free Turbo SQL scripts improve the performance of DNN significantly. Sebastian was awarded as a Microsoft Most Valuable Professional since 2007.

Brian Dukes has been working with Engage professionally since 2006, but he has been writing code since around 1998. Brian is very passionate about writing code that is easily maintainable and helping others to do the same. He has been a leader in the DNN community and can often be found speaking at conferences and helping others on Twitter, GitHub, and StackOverflow. DNN recognized his community efforts by naming him as a DNN MVP starting in 2012. Outside of work, Brian spends time with his family, serves Jesus at City Lights Church, and supports social justice, fair trade, local, seasonable food, and international adoption.

Credits

PROJECT EDITOR

Kelly Talbot

TECHNICAL EDITORS

Sebastian Leupold

Brian Dukes

PRODUCTION EDITOR

Rebecca Anderson

COPY EDITORS

Kim Heusel

Kezia Endsley

MANAGER OF CONTENT DEVELOPMENT & ASSEMBLY

Mary Beth Wakefield

PRODUCTION MANAGER

Kathleen Wisor

MARKETING DIRECTOR

David Mayhew

MARKETING MANAGER

Carrie Sherrill

PROFESSIONAL TECHNOLOGY & STRATEGY DIRECTOR

Barry Pruett

BUSINESS MANAGER

Amy Knies

ASSOCIATE PUBLISHER

Jim Minatel

PROJECT COORDINATOR, COVER

Brent Savage

PROOFREADER

Kim Wimpsett

INDEXER

Johnna VanHoose

COVER DESIGNER

Wiley

COVER IMAGE

©Getty Images/Martin Barraud

Acknowledgments

We would like to thank Wrox Press for its continued commitment to the DNN open source community, DNN Corp for its sponsorship of this book, the Wrox editors who worked with us throughout the process to get this book over the finish line, and our friends and families for their support and patience.

Try Safari Books Online FREE for 15 days and take 15% off for up to 6 Months*

Gain unlimited subscription access to thousands of books and videos.



With Safari Books Online, learn without limits from thousands of technology, digital media and professional development books and videos from hundreds of leading publishers. With a monthly or annual unlimited access subscription, you get:

- Anytime, anywhere mobile access with Safari To Go apps for iPad, iPhone and Android
- Hundreds of expert-led instructional videos on today's hottest topics
- Sample code to help accelerate a wide variety of software projects
- Robust organizing features including favorites, highlights, tags, notes, mash-ups and more
- Rough Cuts pre-published manuscripts

START YOUR FREE TRIAL TODAY!

Visit: www.safaribooksonline.com/wrox

*Discount applies to new Safari Library subscribers only and is valid for the first 6 consecutive monthly billing cycles. Safari Library is not available in all countries.



An Imprint of **WILEY**
Now you know.

Safari 
Books Online



Programmer to Programmer™

Connect with Wrox.

Participate

Take an active role online by participating in our P2P forums @ p2p.wrox.com

Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble

Join the Community

Sign up for our free monthly newsletter at newsletter.wrox.com

Wrox.com

Browse the vast selection of Wrox titles, e-books, and blogs and find exactly what you need

User Group Program

Become a member and take advantage of all the benefits

Wrox on **twitter**

Follow @wrox on Twitter and be in the know on the latest news in the world of Wrox

Wrox on **facebook**

Join the Wrox Facebook page at facebook.com/wroxpress and get updates on new books and publications as well as upcoming programmer conferences and user group events

Contact Us.

We love feedback! Have a book idea? Need community support?
Let us know by e-mailing wrox-partnerwithus@wrox.com

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.