Chapter 7
"Custom Module Development"
In this package, you will find:

A Biography of the authors of the book
A preview chapter from the book, Chapter 7 "Custom Module Development”
A synopsis of the book’s content
Information on where to buy this book

About the Author

Daniel Egan

Daniel Egan has held a variety of positions in the information technology and engineering fields over the last nine years. Currently, he is a System Development Specialist for Automated Data Processing's Southern California region, working extensively in database applications and web development. Daniel is an MCPand MCSD.

In addition to his development work, he teaches a VB.NET Certification course at California State University, Fullerton as well as serves on its .NET Advisory board. He is also the founder and chief author of Dot Net Doc (www.DotNetDoc.com), a .NET and DNN developer resource website built using the DotNetNuke framework. He has written numerous articles on DotNetNuke and the underlying DNN architecture. He is also the founder of the LA/Orange County DNN Usergroup and is currently working on two DNN-related projects: DNNUsergroup Online (www.DNNUGOnline.com), a portal designed to allow usergroups to broadcast their meetings online, and DotNetNuke Radio, a live internet radio show about DotNetNuke.

Michael Washington

Michael Washington is a website developer and an ASP.NET, C#, and Visual Basic programmer. He is a DotNetNuke Core member and has been involved with DotNetNuke for over three years. He is the author of numerous DotNetNuke modules and tutorials. He is one of the founding members of the Southern California DotNetNuke Users group (www.socaldug.org). He has a son, Zachary, and resides in Los Angeles with his wife Valerie.

Steve Valenzula

Steve Valenzula is the manager of the University Extended Education (UEE) IT Department at California State University, Fullerton, where he has worked for the last five years. Steve has worked specifically with DotNetNuke for over two years, in that time re-designing and delivering various Extended Education websites on the DotNetNuke portal framework as well as designing and delivering custom modules that support the function of University Extended Education.
About the Reviewers

Jerry Spohn

Jerry Spohn has been working with computers since the age of 11, at which he first began learning programming on a Commodore VIC 20. Times have changed, and he moved through the interesting world of IBM mainframes into PCs. After taking numerous courses on database design, programming, and object-oriented methodologies, he moved into Visual Basic and other Microsoft languages.

Jerry currently works as a Development Manager for a medium-sized software company in Pennsylvania. He also manages over 25 different websites using DotNetNuke, and is the owner of Spohn Software LLC, which does custom development across the entire Microsoft development toolset.

Jim Wooley

Jim Wooley began working on portals by building his own engine base on XML and XSLT. Just as he was about to release it, the IBuySpy Portal was released.

Promptly dumping his custom solution, he has been working on extending and deploying a number of IBuySpy and DotNetNuke portals. He is always striving to stay at the forefront of technology and enjoys the thrill of a new challenge. In addition, he attempts to pass on the insights he has gained by being active in the community, including leading the Atlanta VB Study Group and serving as INETA NorAm Membership Manager for the Georgia region.
Building Websites with VB.NET and DotNetNuke 4

DotNetNuke is a free, open-source evolution of Microsoft's celebrated ASP.NET reference implementation, the IBuySpy portal solution kit. DotNetNuke began life as a framework for constructing data-driven intranet and Internet portal applications, and has now developed into an advanced web content management system with tools to manage a dynamic and interactive data-driven website. The DotNetNuke portal framework allows you to quickly create a fully featured community-driven website, complete with standard modules, user registration, and integrated security. This free open-source application puts a staggering range of functionality into your hands, and, either by using it as is or by customizing it to your requirements; you are giving your projects a great head start.

Supported and tested by thousands of developers in the DotNetNuke community across the world, the DotNetNuke framework, on one hand, offers you the luxury of a well-tested and proven architecture, and on the other, the ability to manage your site through an easy web-based administration system.

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

What This Book Covers

Chapter 1 introduces DotNetNuke (DNN) and discusses the meaning and purpose of web portals, and the common aspects of successful web portals. It looks at different types of open-source web portals, and discusses why we selected DotNetNuke for this book. We then meet our fictional client Coffee Connections and, using user stories, gather the requirements needed to build this client's site.

In Chapter 2 we see how to install a local version of DotNetNuke with Microsoft SQL Server and SQL Server 2005 Express, and cover setting the required permissions on your machine to run DNN properly.

In Chapter 3 we cover users, roles, and pages. Users are the individuals who visit or administer your portal, and their power depends on the roles that they have been assigned. We discuss how each page of your portal can be administered differently, laying the foundation for the rest of the book. From defining users, to registration, to security roles, this chapter will help you to begin administering a DNN portal.

In Chapter 4 we cover the standard modules that come pre-packaged with DotNetNuke. We cover their basic uses as well as situations they may be used in. You will use these modules to build your portal's content.

Chapter 5 introduces the administrative functions available to the host and admin logins. These are special logins that have access to all areas of your portal, and are used to secure your site and make changes to its content. This chapter takes you through the tools to make sure you are
comfortable with all that is available to you. Understanding the core architecture of DNN is essential if you want to extend the system or even modify the existing code.

In Chapter 6 we learn how the DotNetNuke framework builds the pages, and the major classes that drive it.

In Chapters 7 and 8 we take the knowledge we learned in the last chapter and use it to build a custom module. You will learn everything you need to know to start building your own modules so you can extend the capabilities of your portal. After creating your user controls, you will create your data access and business logic layers.

In Chapter 8 you will learn about the DotNetNuke Data Access Layer (DAL) and the DAL+, which take much of the routine work out creating custom modules. We finish our look at development by seeing how to package your module for distribution.

Chapter 9 talks about skins. A skin is the outer layer of your site, and defines the look and feel of the portal. In this chapter we design a custom skin for the Coffee Connections site. You will learn the skills needed to skin both your portal and your module containers.

When you finally have your portal the way you want it to look and function, you are ready to deploy it, and that is what Chapter 10 shows you how to do. The chapter advises on what you should look for in a web host and helps to steer you clear of common deployment mistakes.

In Chapter 11 we show you how to take advantage of one of the most exciting features of DotNetNuke: multiple portals. These are additional portals that use the same underlying database, but can contain different content. So instead of just having one website, you can create as many as you need using just one DotNetNuke installation. From parent portals to child portals, this chapter gives you the information necessary to create new portals from scratch or to use the new template structure built into the framework.
Custom Module Development

In this chapter, we will be creating a custom module for the Coffee Connections portal. A custom module can consist of one or more custom web controls. The areas that will be covered are:

- Setting up the development environment
- Creating a "Hello World!" View Control
- Creating a "Hello Edit" Edit control

Setting up the Development Environment

To develop modules for DotNetNuke you must first have a DotNetNuke installation running on the computer on which you intend to develop them.

DotNetNuke comes in two versions, a source version, and an install version. The install version is also packaged as the DotNetNuke Starter Kit. They are functionally the same. Surprisingly, it is recommended that you use the install version to develop modules. The reason for this is that the source version should only be used if you intend to change the DotNetNuke core code. This is not recommended as it may not allow you to upgrade your installation in the future.

In this chapter, Visual Web Developer Express 2005 will be used (hereafter referred to as Visual Studio); however, the instructions are the same for all versions of Visual Studio 2005.

Download and install the DotNetNuke Starter Kit. Then open Visual Studio and from the File menu select New Web Site.

Select the DotNetNuke Web Application Framework template that was installed by the DotNetNuke Starter Kit. Ensure that the Location is set to File System.

Follow the directions on the page that will appear after installation to complete any configuration and launch your DotNetNuke website.

DotNetNuke is constantly changing as it evolves and the best way to get up-to-date help and information is to use the DotNetNuke message board at DotNetNuke.com. There are installation documents available at DotNetNuke.com that you can download that will assist you.

The Coffee Shop Listing Module

In this chapter, we will go through the process of creating a DotNetNuke module from top to bottom. Although we will go through quite a bit of code in this chapter, we do not cover every single line. To help you as you work through the chapter, the complete source code is available for download from the publisher’s site (http://www.PacktPub.com/support).

One of the main attractions for the Coffee Shop Listing module is that users will be able to search, by zip code, for coffee shops in their area. After searching, the users will be presented with the shops in their area. This will be accomplished using the View control.

In addition, the module will be configurable to allow certain users to add coffee shops using the Edit control.

The administrator of the site will be able to configure which users are allowed to add coffee shops using the Settings control.
To allow the focus of this chapter to be on module development, we will not spend time on validation of the various controls, instead we will focus only on what is necessary to create the module.

Creating the View Control

Our first example will be the traditional Hello World! example. Later, we will alter this control to complete the Coffee Shop Listing module.

1. Open Visual Studio and select File from the toolbar, then click Open Web Site.

2. Next, select the root directory of the DotNetNuke website and click the Open button.
The website will open and display in the **Solution Explorer** window.
3. To ensure that your development environment is configured properly, from the toolbar select View, then click Output to display the output window, then from the toolbar select Debug, and then click Start Without Debugging.

You might have a long wait while Visual Studio builds the website. When the build is complete the output windows should show no errors (failed should be 0).
When the build is complete the website will automatically launch.


5. Name the folder CoffeeShopListing
6. Next, right-click on the CoffeeShopListing folder and select Add New Item.

7. When the Add New Item menu appears, select the Web User Control and enter ShopList.ascx in the Name box and check the box next to Place code in separate file. Also, ensure that Visual Basic is selected in the Language dropdown.

The ShopList.ascx file will now appear under the CoffeeShopListing folder.

The source for the ShopList.ascx will also appear in the main window. If it is in the design view, click the Source button in the lower left-hand corner of the window to switch to the Source view.
8. Replace all the code with this code:

```vbnet
<%@ Control language="vb" AutoEventWireup="false"
       Inherits="EganEnterprises.CoffeeShopListing.ShopList"
       CodeFile="ShopList.ascx.vb"%>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

You will see wavy blue lines that indicate errors. These errors will be cleared up when we replace the code in the code-behind file.


10. Replace all the code with the following code:

```vbnet
Imports DotNetNuke
Imports System.Collections.Generic

Namespace EganEnterprises.CoffeeShopListing
    Partial Class ShopList
        Inherits Entities.Modules.PortalModuleBase
        Protected Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
            Try
                Label1.Text = "Hello World!"
            Catch exc As Exception
                Exceptions.ProcessModuleLoadException(Me, exc)
            End Try
        End Sub
    End Class
End Namespace
```

11. Select **File** then **Save All** to save the changes. Now, from the toolbar select **Build**, then click on **Build Page**.

![Build Page](image)

If you have build errors it is best to download the code using the link noted at the beginning of the chapter and compare it to your own.

**Displaying the Module**

We will now walk through the steps needed to configure the module and view it in your DotNetNuke website.

1. In Visual Studio, select **Debug** then **Start Without Debugging**. When the site comes up click the **Login** link.

![Login](image)

2. Log in as the **host** user.

![Login Form](image)

The password for the host account is usually **dnnhost**. However, refer to the first page of your DotNetNuke website as this may change.

3. Click on the Host menu and select Module Definitions.

4. From the Module Definitions menu (click on the small black down-pointing arrow in the upper left-hand corner) select Add New Module Definition.

5. When the Module Definition screen comes up:
   - Enter CoffeeShopListing for Module Name.
   - Enter CoffeeShopListing for Folder Name.
   - Enter CoffeeShopListing for Friendly Name.
   - Enter CoffeeShopListing for Description.
   - Enter 01.00.00 for Version.
   - Click the Update button.
6. Near the bottom of the **Edit Module Definitions** form, enter **CoffeeShopListing** in the **New Definition** box and click the **Add Definition** link.

7. Click the **Add Control** link.

8. When the **Edit Module Control** form appears:
   - Enter **CoffeeShopListing** for **Title**.
   - Use the dropdown to select **DesktopModules/CoffeeShopListing/ShopList.ascx** for **Source**.
Chapter 7

- Use the dropdown to select **View** for **Type**.
- Click the **Update** button.

![Edit Module Control](image)

9. On the **Admin** menu select **Pages**.

![Admin menu with Pages highlighted](image)

10. On the Pages form click **Add New Page**.

![Add New Page button](image)

11. On the **Edit Page** form:
   - Enter **Coffee Shop Listing** for **Page Name**.
   - Enter **Coffee Shop Listing** for **Page Title**.
   - Enter **Coffee Shop Listing** for **Description**.
   - Check the box for **All Users** under **View Page**.
   - Then click the **Update** button.
The **Coffee Shop Listing** tab will now appear on the toolbar.

12. Click on it to navigate to that page.

13. From the administration bar at the top of the site, select **CoffeeShopListing** from the **Module** dropdown and click **Add**.

The module will now appear.
Next, we will intentionally generate an error to explore the functionality of the Log Viewer. Leave the website open and return to Visual Studio. In Visual Studio, add the following, highlighted line into your code as shown:

```vbnet
Label1.Text = "Hello World!"
Throw New Exception("Something didn't work right.")
Catch exc As Exception
    Exceptions.ProcessModuleLoadException(Me, exc)
End Try
```

Save the page and return to the site in the web browser and click on the Coffee Shop Listing tab to refresh the page. The page will now show an error message.

Select Log Viewer from the Admin menu. Locate the most recent entry for Module Load Exception and click on it to expand it. We can see it indicates that the error is in the ShopList.ascx.vb file and indicates the line number.
You may see multiple General Exceptions errors for `System.IO.Path.NormalizePathFast`. To prevent this, develop DotNetNuke in a much shorter physical path folder to reduce the total number of characters that compose the path and filename (for instance `C:\DotNetNuke`).

Return to Visual Studio and remove the `Throw New Exception("Something didn't work right.")` line. Save the page and return to the site in the web browser and click on the Coffee Shop Listing tab to refresh the page.

**What we have Accomplished**

We have just explored a few core concepts of DotNetNuke module development:

- The DotNetNuke module folder structure
- Inheriting from PortalModuleBase
- Module configuration
- Diagnosing errors using the Log Viewer

**The Module Folder Structure**

A DotNetNuke module is made up of User Controls and their associated code-behind files that reside in folders in the `DesktopModules` directory. Optionally, other code files for the module that are not associated with a User Control (for example, Data Access Layer and Business Logic Layer code) reside in the `App_Code` directory. In later steps we will create code that will reside in the `App_Code` directory.
Inheriting from `PortalModuleBase` is essential because it is the base class for all User Controls in DotNetNuke. Using the base class is what gives our control consistency in its appearance with the portal it resides in, and provides functionality such as the menu and portal access security.

This class also gives us access to useful items such as the current user and the current `ModuleId`. In later steps you will see how these items allow the User Control to interact with the DotNetNuke framework to provide most of the functionality you would desire.

**Module Configuration**

Adding module definitions makes the module appear in the control panel module dropdown when you are signed on as host or admin. It connects your controls to the portal framework. In the walk-through we configured the module to have one User Control. In later steps, we will create and configure two additional User Controls.

**Diagnosing Errors using the Log Viewer**


```
Catch exc As Exception
    Exceptions.ProcessModuleLoadException(Me, exc)
```

This is useful during module development as well as to assist an administrator in diagnosing problems when the module is deployed to production.

**Navigation and Localization**

At this stage you can see that creating a module from a User Control is relatively straightforward. Programming multiple User Controls to interact with each other is not as straightforward. The next walk-through will be devoted to this subject as well as demonstrating *Localization*, which allows your module to display its text in multiple languages.

We will create an Edit User Control. For now it will just say *Hello Edit!* Later we will alter it to complete the Coffee Shop Listing module.

**Create EditShopList.ascx**

1. In Visual Studio, right-click on the `CoffeeShopListing` folder and select *Add New Item*.
2. When the **Add New Item** menu comes up, select **Web User Control** and enter `EditShopList.ascx` in the **Name** box. Ensure that the **Place code in separate file** box is checked and click the **Add** button.

3. When the page comes up in source view, replace all the code with the following code:

```csharp
<%@ Control language="vb" AutoEventWireup="false" 
Inherits="EganEnterprises.CoffeeShopListing.EditShopList" 
CodeFile="EditShopList.ascx.vb"%>
<br />
<br />
<asp:Label id="Label1" runat="server" Text="Label"></asp:Label>
<br />
<asp:LinkButton id="cmdReturn" runat="server" Text="Return" 
BorderStyle="none" CssClass="CommandButton" 
CausesValidation="False"></asp:LinkButton>
```

4. Right-click on `EditShopList.ascx` in the Solution Explorer and select **View Code**. When the source code is displayed, replace all the code with the following code:

```csharp
Imports DotNetNuke
Namespace EganEnterprises.CoffeeShopListing
    Partial Class EditShopList
        Inherits Entities.Modules.PortalModuleBase
        Protected Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
            Label1.Text = "Hello Edit!"
        End Sub
        Protected Sub cmdDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdReturn.Click
            ' Redirect back to the portal
            Response.Redirect(NavigateURL())
        End Sub
    End Class
End Namespace
```

5. Save and build the page.
Navigation

The EditShopList.ascx page doesn't do much at this point. For now, it is being used to demonstrate how to navigate between User Controls (or pages) of your module. Now we will place a link and a menu item on the ShopList page that will navigate to the EditShopList page.

1. Right-click on ShopList.ascx and select View Code.
2. When the source code is displayed, add the highlighted line as shown:
   ```csharp
   Inherits Entities.Modules.PortalModuleBase
   Implements Entities.Modules.IActionable
   ``
3. Next, insert the following code above the line: End Class.
   ```csharp
   #Region "Optional Interfaces"
   Public ReadOnly Property ModuleActions() As _
   ModuleActions
   Get
   Dim Actions As New _
   Actions.Add(GetNextActionID, Localization.
   ModuleActionType.AddContent, "", EditUrl(), False,
   Return Actions
   End Get
   End Property
   #End Region
   ``
4. Save and build the page.

Localization

In the final step for this section we will create a resource file that can be used to replace the text of one of the links (for example to change the language). For now the link will be in English and we will only localize one link. Normally you would localize all the text elements of your module so that it can be used in any language.
1. Right-click on the CoffeeShopListing folder and select **Add ASP.NET Folder** and then **App_LocalResources**.

![Add ASP.NET Folder](image)

2. When the **App_LocalResources** folder appears, select **Add New Item**.

![Add New Item](image)

3. When the **Add New Item** box comes up, select **Resource File** as the template, enter **ShopList.ascx.resx** in the **Name** box, and click the **Add** button.

![Add New Item - C:\Documents and Settings\My Documents\Visual Studio 2005\Web Sites\DotNetN...](image)
4. When the resource editor window appears, enter `AddContent.Action` in the **Name** column and **Add Coffee Shop** in the **Value** column.

![Image of resource editor window]

5. Save the file and close it.

**Update the Configuration**

In Visual Studio, press `Ctrl+F5` to start the DotNetNuke site.

1. Log in as host.
2. From the **Host** menu select **Module Definitions**.
3. From the **Module Definitions** click the edit symbol next to the **CoffeeShopListing** module to select it.
4. Click the **Add Control** link toward the bottom of the **CoffeeShopListing** module definition.
5. When the control configuration screen appears, configure it with the following settings:
   - Enter **Edit** for **Key**.
   - Enter **Edit Shoplist** for **Title**.
   - Use the dropdown to select `DesktopModules/CoffeeShopListing/EditShopList.ascx` for **Source**.
   - Use the dropdown to select **Edit** for **Type**.
   - Click the **Update** button.
Navigate from ShopList to EditShopList

Next, click the Coffee Shop Listing link on the menu bar in the DotNetNuke site. On the CoffeeShopListing module, you will see that there is now a link to Add Coffee Shop.

Click on the link and you will navigate to the Edit page.
Click the **Return** button and you will navigate back to the ShopList page. Click the menu icon in the corner of the **Coffee Shop Listing** module and notice that there is also an **Add Coffee Shop** link there.

## What we have Accomplished

We have explored a few additional concepts of DotNetNuke module development:

- IActionable
- NavigateURL
- Localization

### IActionable

We only implemented the `IActionable` interface in one place in the code yet the **Add Coffee Shop** link shows up in two places on the module. It shows up on the ShopList page and on the modules menu. This demonstrates the benefit of the DotNetNuke API. The link and the menu item not only show up in standardized places, but they also show up based on the security roles that you indicate. For example, currently the **Add Coffee Shop** link will only show up if you are logged in as the host or administrator account.

To add an action menu item to the module actions menu, we create an instance of a `ModuleActionCollection`. This is done in the `ModuleActions` property declaration.

```vbnet
Public ReadOnly Property ModuleActions() As _
Get
    Dim Actions As New _
```

We then use the `Add` method of this object to add an item to the menu.

```vbnet
Actions.Add(GetNextActionID, _
    Localization.GetString( _
        Entities.Modules.Actions.ModuleActionType.AddContent, _
        LocalResourceFile), _
```
Custom Module Development

```
Entities.Modules.Actions.ModuleActionType.AddContent, _,
"", _,
"", _
EditUrl(), _
False, _
True, _
False)
Return Actions
End Get
End Property
```

The parameters of the `Actions.Add` method are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>The <code>GetNextActionID</code> function (found in the <code>ActionsBase.vb</code> file) will retrieve the next available ID for your <code>ModuleActionCollection</code>. This works like an auto-increment field, adding one to the previous action ID.</td>
</tr>
<tr>
<td>Title</td>
<td>String</td>
<td>The title is what is displayed in the context menu form your module.</td>
</tr>
<tr>
<td>CmdName</td>
<td>String</td>
<td>If you want your menu item to call client-side code (JavaScript), then this is where you will place the name of the command. This is used for the delete action on the context menu. When the delete item is selected, a message asks you to confirm your choice before executing the command. For the menu items we are adding we will leave this blank.</td>
</tr>
<tr>
<td>CmdArg</td>
<td>String</td>
<td>This allows you to add additional arguments for the command.</td>
</tr>
<tr>
<td>Icon</td>
<td>String</td>
<td>This allows you to set a custom icon to appear next to your menu option.</td>
</tr>
<tr>
<td>URL</td>
<td>String</td>
<td>This is where the browser will be redirected to when your menu item is clicked. You can use a standard URL or use the <code>EditURL</code> function to direct it to another module. The <code>EditURL</code> function finds the module associated with your view module by looking at the key passed in. These keys are entered in the Module Definition.</td>
</tr>
<tr>
<td>ClientScript</td>
<td>String</td>
<td>As the name implies, this is where you would add the client-side script to be run when this item is selected. This is paired with the <code>CmdName</code> attribute above. We are leaving this blank for your actions.</td>
</tr>
</tbody>
</table>
### Parameter Type Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseActionEvent</td>
<td>Boolean</td>
<td>This determines if the user will receive notification when a script is being executed.</td>
</tr>
<tr>
<td>Secure</td>
<td>SecurityAccessLevel</td>
<td>This is an Enum that determines the access level for this menu item.</td>
</tr>
<tr>
<td>Visible</td>
<td>Boolean</td>
<td>Determines whether this item will be visible.</td>
</tr>
<tr>
<td>NewWindow</td>
<td>Boolean</td>
<td>Determines whether information will be presented in a new window.</td>
</tr>
</tbody>
</table>

**NavigateUrl**

If you want to create a link that you do not want to appear as a menu item, you can simply use code such as `Response.Redirect(NavigateURL());`.

The `NavigateURL` function works in conjunction with DotNetNuke URL rewriting. URL rewriting is a function DotNetNuke performs to create URL's that are more easily indexed by search engines.

If we run the code in Debug mode we can see that `NavigateURL()` resolves to `http://localhost:1545/DotNetNuke/CoffeeShopListing/tabid/53/Default.aspx`. Your instance will certainly have a different URL, but the result will be the same that directs the user back to the default page of your module, which in this case is the ShopList User Control.

![Visual Studio object browser showing NavigateURL() method with multiple overloads](image)

We can see in the Visual Studio object browser that the `NavigateURL()` method has multiple overloads.
Adding Localization

You will notice that the second parameter of the `Add()` method in the `IActionable` interface asks for a title. This is the text that will be displayed. In our code you will notice that instead of using a string, we use the `Localization.GetString()` method to get the text from the local resource file.

```csharp
Actions.Add(GetNextActionID, _
            Localization.GetString( _
            Entities.Modules.Actions.ModuleActionType.AddContent, _
            LocalResourceFile), _
            Entities.Modules.Actions.ModuleActionType.AddContent, _
            "", _
            "", _
            EditUrl(), _
            False, _
            True, _
            False)
```

This allows the portal administrator to set the language of their choice by simply replacing the resource file. For further discussion of localization refer to the localization document available at DotNetNuke.com.

Summary

The module is not complete, however, we have covered many important concepts that you will most likely use in every module you create. Essentially a DotNetNuke module is made up of web controls that inherit from `Entities.Modules.PortalModuleBase`. Navigation and localization were covered because their proper use will allow you to create modules that integrate well into your portal. In addition, we also covered exception handling that will aid you in your module development.

In the next chapter we will cover connecting to the database. We will also cover optional interfaces that will allow you to import and export module data and to integrate the module into the DotnetNuke portal search.
Where to buy this book


Free shipping to the US, UK, Europe, Australia, New Zealand and India.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.