DotNetNuke 4.0 Module Developers Guide (Part 1)

Michael Washington

Version 1.0.0
Last Updated: January 3, 2007
Category: DotNetNuke v4.3.7

DotNetNuke 4.0 Module Development

Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results of the use of this document remains with the user.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Perpetual Motion Interactive Systems, Inc. Perpetual Motion Interactive Systems may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Perpetual Motion, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

DotNetNuke® and the DotNetNuke logo are either registered trademarks or trademarks of Perpetual Motion Interactive Systems, Inc. in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

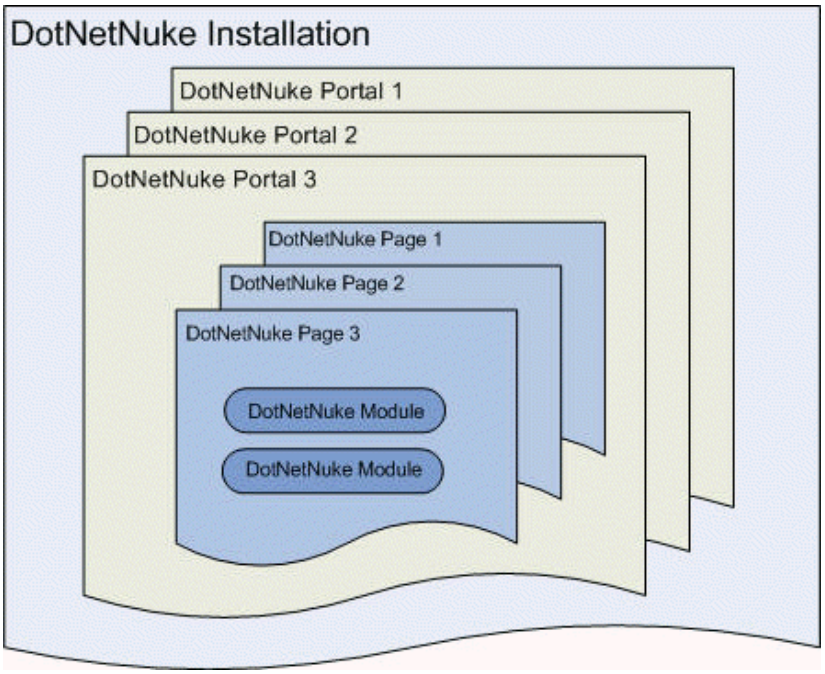
DotNetNuke 4.0 Module Development

Abstract

This guide covers module development for the DotNetNuke framework, covering DotNetNuke 4 (ASP.NET 2.0)

Contents

Introduction to DotNetNuke Module Development



What is a DotNetNuke Module?

In order to understand what a DotNetNuke module is, it is important to first understand what DotNetNuke is.

DotNetNuke is a Framework

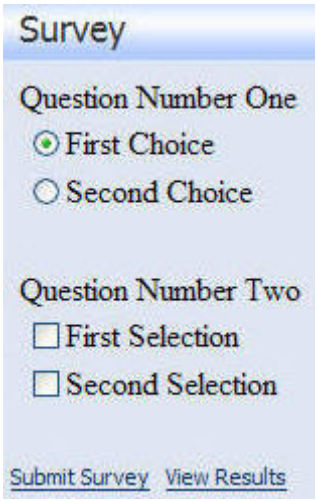
DotNetNuke is a program that runs on Microsoft ASP.NET. It is also a framework, meaning, it is a program that is designed to be extended. One of the ways you extend the framework is to create modules. These modules are installed inside a DotNetNuke

installation and when they run in that DotNetNuke installation they extend the framework to create a DotNetNuke website also called a portal.

A single DotNetNuke installation will allow the creation of thousands of individual portals (as much as the server hardware can handle). DotNetNuke portals are configured to display pages and the pages are configured to display modules.

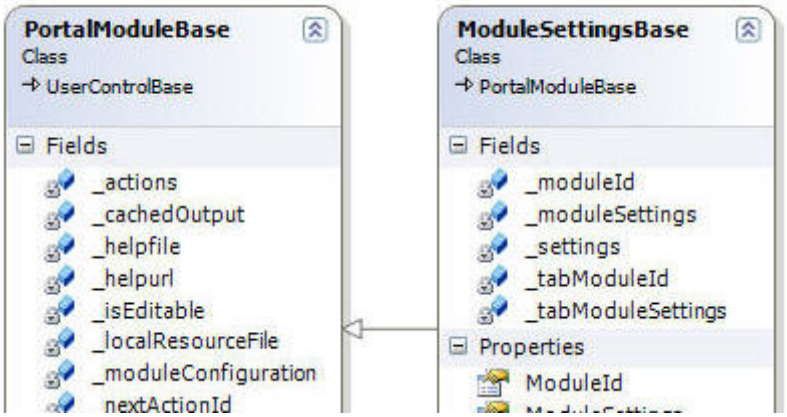
The portal is accessed by users via a web browser (this can be over the internet or an intranet). These users can be either anonymous visitors or users with accounts and passwords. The modules that are installed and configured in the DotNetNuke installation provide the functionality that the users can perform.

For example, the Survey module allows visitors to participate in a survey and possibly see the tabulated results.



The DotNetNuke framework provides the various functions such as determining if a user is logged in, displaying the look and feel, determining what is on a page and who can see it. The Survey module extends the framework and allows users to participate in the survey.

The developer of the Survey module needs only to create the survey functionality. The DotNetNuke framework handles all the required elements to access the module including the installation of the module. The Survey module developer does this by writing code that interfaces with the DotNetNuke API (Application Programming Interface). This is done by creating user controls that inherit from the DotNetNuke base classes (*PortalModuleBase* or *ModuleSettingsBase*)



DotNetNuke Host and Administrator Accounts

A Host account is the administrator of all portals of a DotNetNuke installation. The Host account is also used for installing modules in the installation. In addition, the Host account creates and configures Administrator accounts. Administrator accounts only administer a particular portal. An Administrator account cannot install (or uninstall) modules but is able to configure the modules that the Host account has made available. The Administrator also indicates what pages the module will be on and which users can access it.

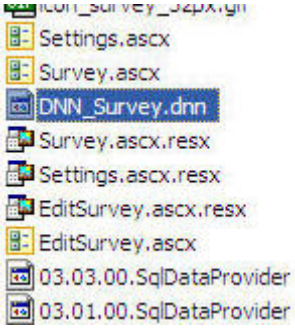
Installing a Module

To better understand how DotNetNuke and modules work, you can examine the installation and configuration of the Survey module.

First you obtain an installation file. You can obtain the installation file for the Survey module from the Survey project page on DotNetNuke.com.

When you download the file you can see that it is in the form of a ".zip" file.

When you open up the ".zip" file you can see that it contains all the elements it needs as well as a "*.dnn*" configuration file.



When you open the "*.dnn*" configuration file you can see that it contains the configuration settings for all the module elements.

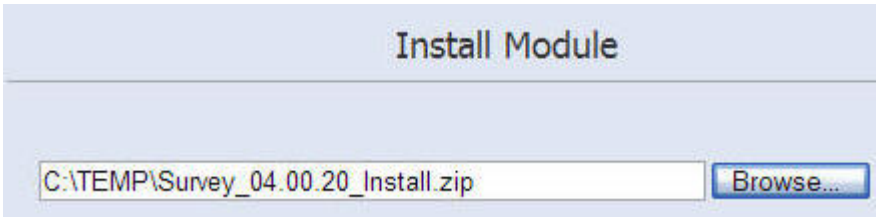
```
<dotnetnuke version="3.0" type="Module">
  <folders>
    <folder>
      <name>DNN_Survey</name>
      <friendlyname>Survey</friendlyname>
      <foldername>Survey</foldername>
      <modulename>DNN_Survey</modulename>
      <description>Survey allows you to create cus
      <version>03.03.00</version>
      <businesscontrollerclass>DotNetNuke.Modul
      <modules>
        <module>
          <friendlyname>Survey</friendlyname>
```

Upload the Survey Module

While logged into the DotNetNuke installation as the Host account, select *Module Definitions* from the *Host* menu.

Then select *Install New Module*.

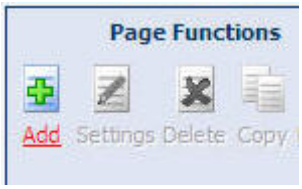
From the *Install Module* page, navigate to the ".zip" file that you downloaded and click the *Save File* link.



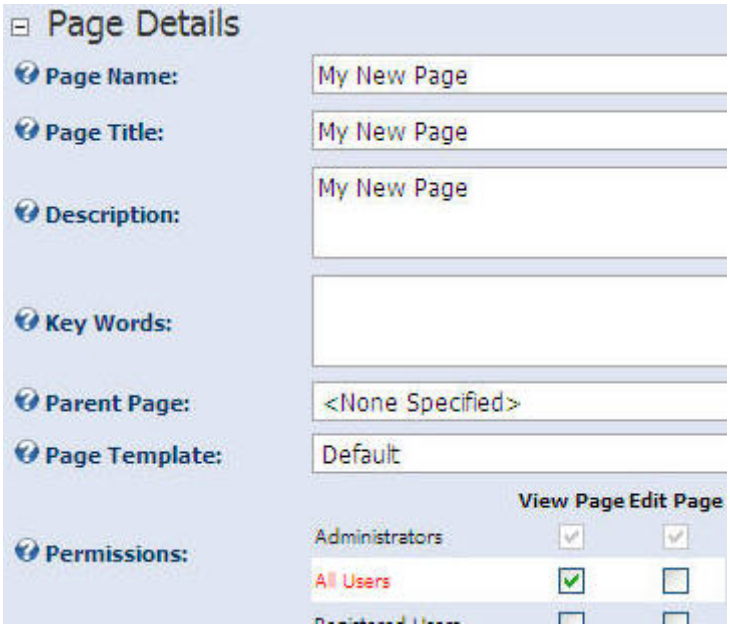
The module will upload...

When you view the installed modules list (*Host* > *Module Definitions*), you will now see the Survey module.

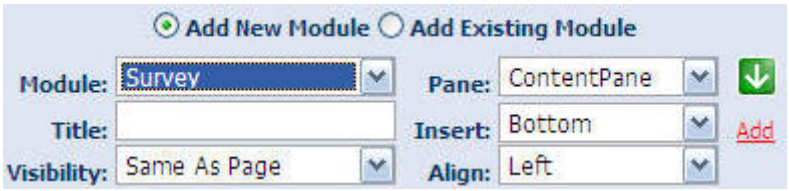
Next you will create a page to place the module on. On the Administrator control panel, click the *Add* link under the *Page Functions* section.



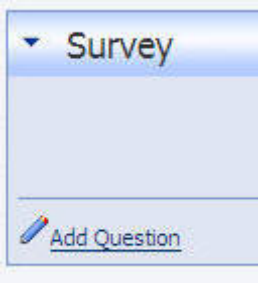
Enter details for the page, ensuring that **All Users** are selected under the *View Page* column. When you have entered the information, click the Update link.



Next, from the Administrator control panel, select *Survey* from the **Module** drop-down list and click the *Add* link.

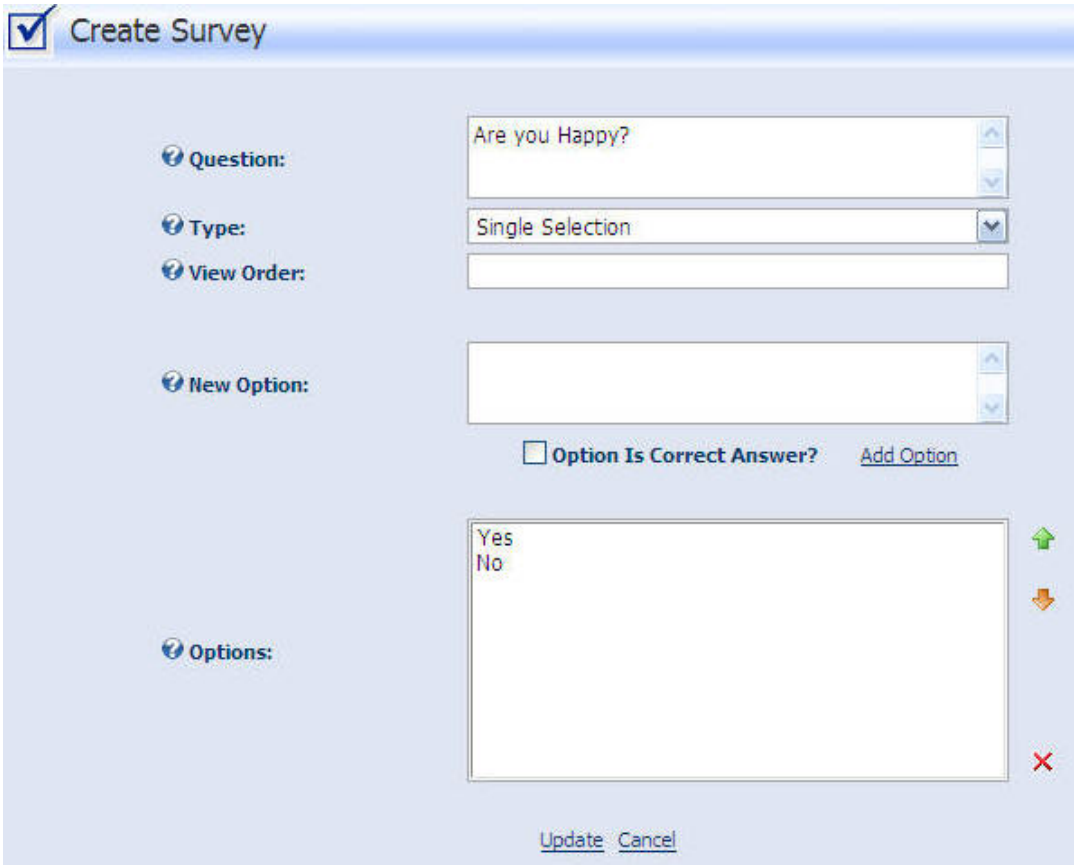


The Survey Module will now appear.



Click the *Add Question* link to configure the module. This link and the functionality that follows is the result of the code contained in the ".zip" package. The preceding functionality is provided by the DotNetNuke Framework.

Create a question and click the *Update* button.

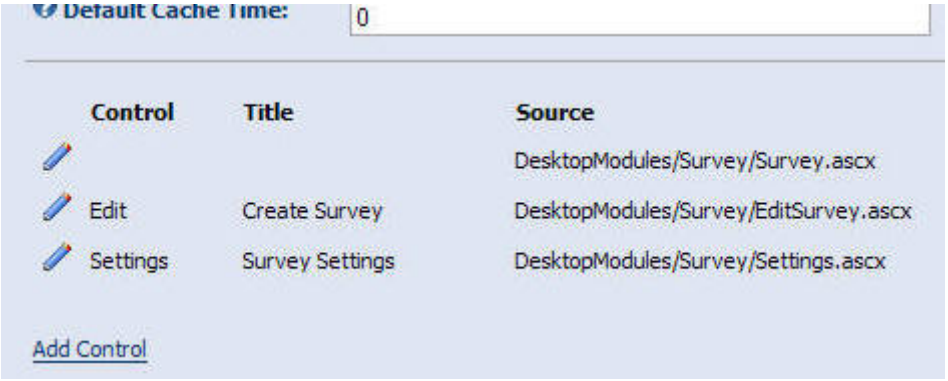


The module will now display.

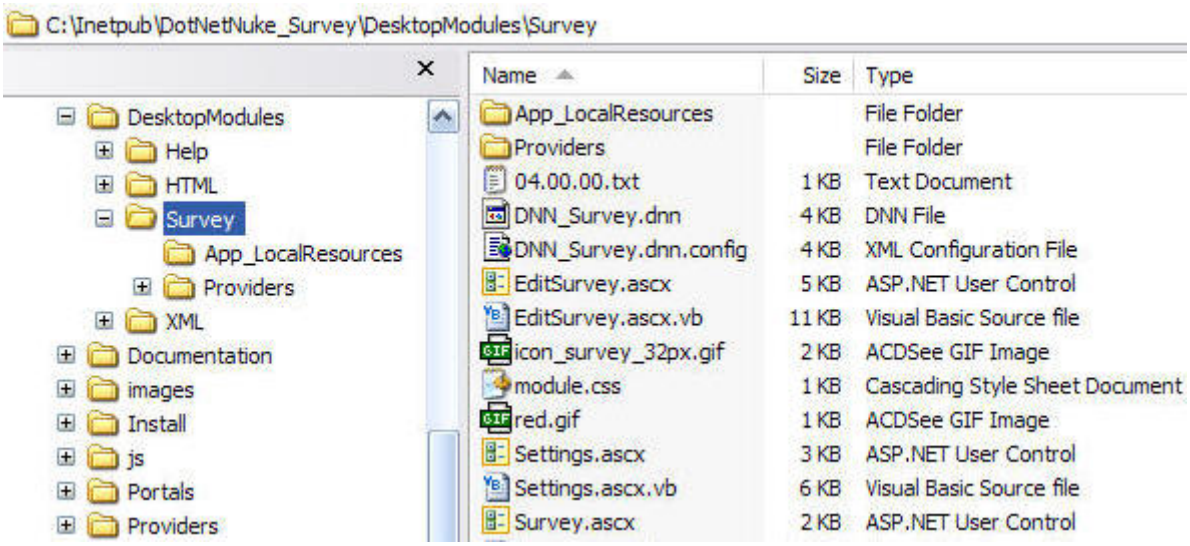
A Look into the DotNetNuke Module Definition

When you go to **Host** > **Module Definitions** and click on the Survey module definition, you can see that there are 3 user controls configured at the bottom of the Module definition:

- DesktopModules/Survey/**Survey.ascx**
- DesktopModules/Survey/**EditSurvey.ascx**
- DesktopModules/Survey/**Settings.ascx**



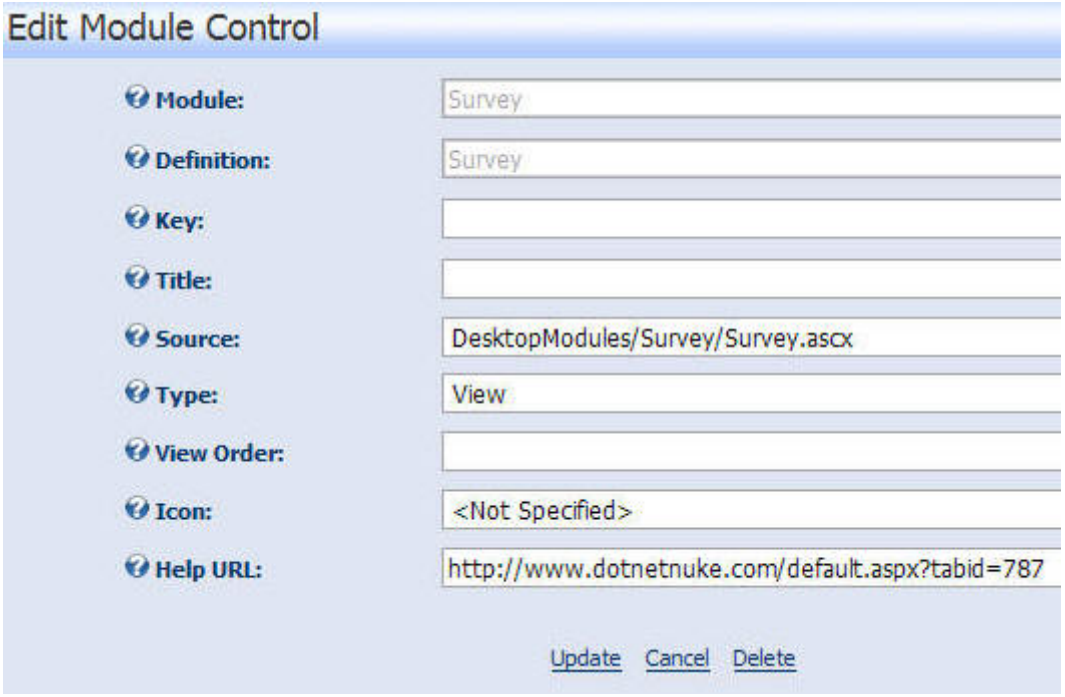
If you look at the *DesktopModules/Survey* directory that was created when you uploaded the Survey module you can see the user controls (and all the other module elements) reside in that directory.



View Control

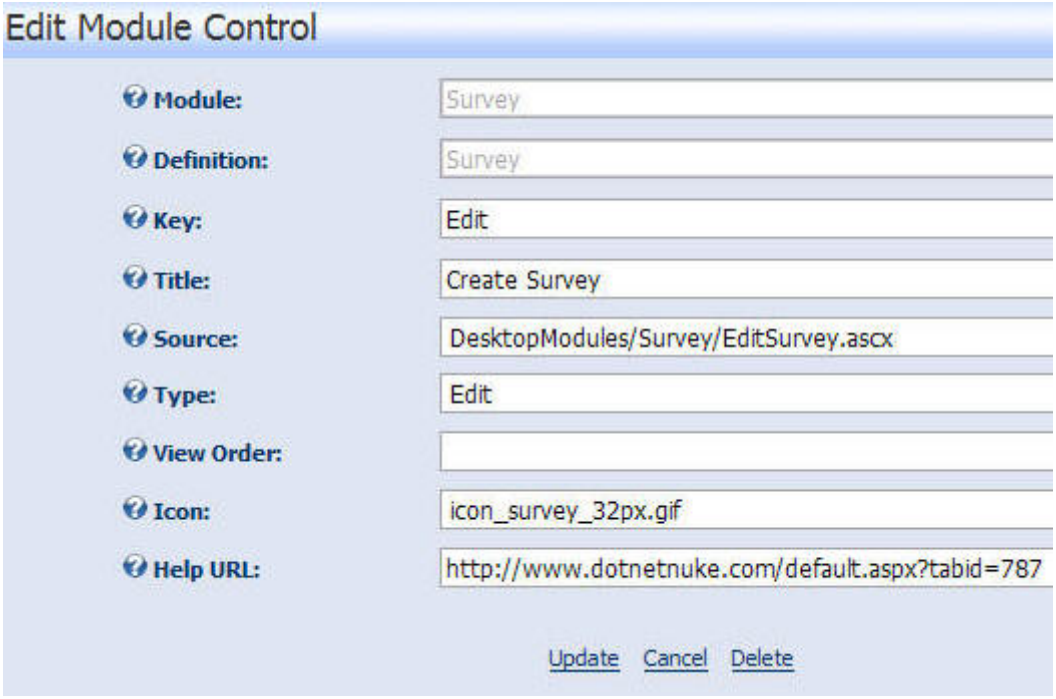
You can click on the pencil icon next to each user control in the module definition to see its details.

When you click the pencil icon next to the *DesktopModules/Survey/***Survey.ascx** control you see that it's **Type** is set to *View*. This is important because it indicates the *security group* that the control will be in. The portal administrator will be able to configure access to each user control based on the **Type**. This control does not have a setting for **Key**. The DotNetNuke framework will therefore display this user control as the default user control.



Edit Control

The *DesktopModules/Survey/***EditSurvey.ascx** control has a **Type** set to *Edit*. This will allow the portal administrator to configure this user control with a different security access than the previous user control. It also has the **Key** configured to *Edit*.



The **Key** is important because it allows the module developer to navigate from the **View control** (*DesktopModules/Survey/***Survey.ascx**) to the **Edit control** (*DesktopModules/Survey/***EditSurvey.ascx**) using code such as this:

```
Response.Redirect(EditUrl())
```

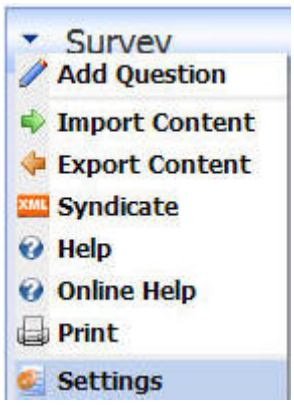
To navigate from the **Edit control** to the **View control** you can use code such as this:

```
Response.Redirect(Globals.NavigateURL(), true)
```

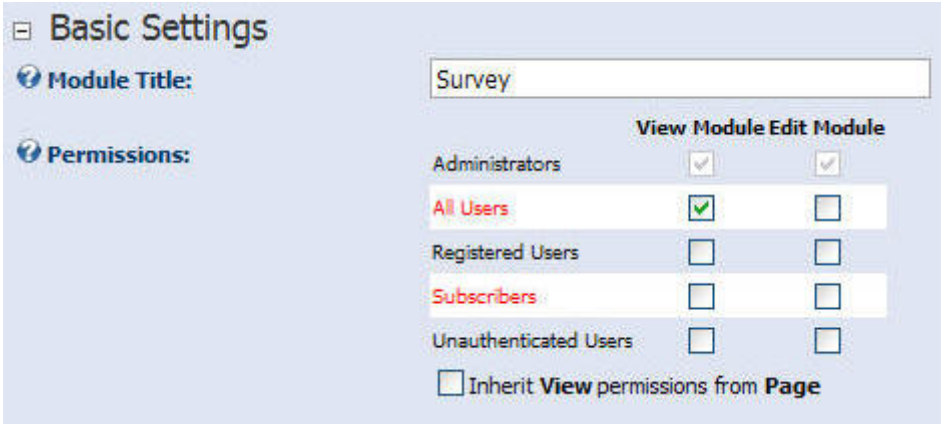
Settings Control

When you look at the details for the **Settings control** *(DesktopModules/Survey/***Settings.ascx**), you can see that its **Type** is set to *Edit* like the **Edit** control, but it's **Key** is set to *Settings*.

This setting together with inheriting from *PortalSettingsBase* and overriding *the LoadSettings() and UpdateSettings()* methods, will instruct the DotNetNuke framework to insert this user control in the **Settings** page for the module. You access this **Settings** page on the module's configuration menu. You access this menu by logging is as the portal administrator (or Host account) and placing the module on a page (if you haven't already done so). Then click the menu link on the module (in the example below it is the small black downward pointing arrow in the upper left hand corner of the module).

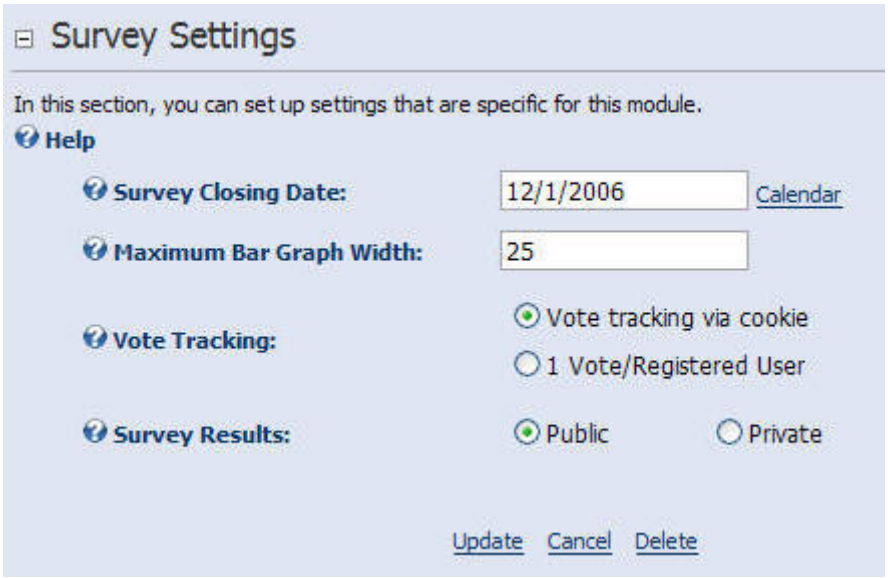


This will bring up the **Settings** page. Here you will be able to configure the security settings for the **View** and **Edit** controls.



To see the contents of the **Settings** user control *(DesktopModules/Survey/***Settings.ascx**), click the plus icon next to **Survey Settings** at the bottom of the settings page.

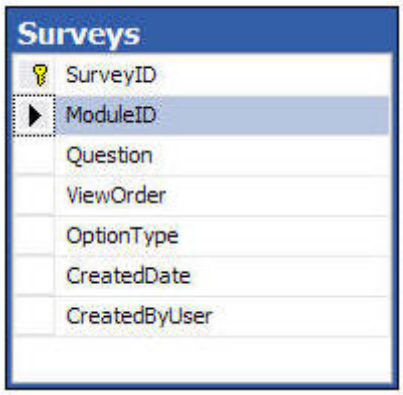
This will allow you to see the contents of the **Settings** user control *(DesktopModules/Survey/***Settings.ascx**).



A Module Can Have Multiple Instances

A portal administrator can add multiple instances of a module to a page. In the picture below, two separate instances of the Survey module have been added to the page.

Therefore, in module development it is important to store the *ModuleID*. You usually would not want the data from one instance of the module to appear in another instance. For the **Survey** module, the *ModuleID* is stored in the Surveys table to properly segment the data.



ModuleID is a unique value across all portals in a DotNetNuke installation. The following code shows how to obtain the current *ModuleId* in a user control that inherits from *PortalModuleBase*:

```
Dim intModuleId As Integer
intModuleId = ModuleId
```

Summary

DotNetNuke is a framework. You implement this framework by extending it's classes and implementing it's interfaces. One way to do this is by creating modules.

For most DotNetNuke is a solution because they use modules that already extend the classes and implement the interfaces. However, in those instances where you are unable to find an existing module that provides the functionality you desire, you can easily create your own modules.

Essentially, a DotNetNuke module is a collection of user controls that are configured to work together. These user controls inherit from either *PortalModuleBase* or *PortalSettingsBase*.

Setting-up Your Development Environment

What You Need:

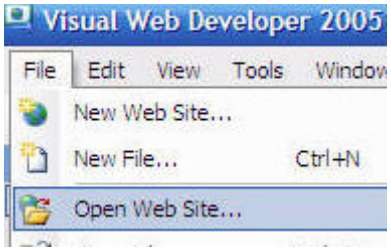
- Visual Studio 2005 or Visual Web Developer Express
- SQL Server Express or SQL Server 2000/2005
- The latest copy of the DotNetNuke "*Install Version*"

Download DotNetNuke

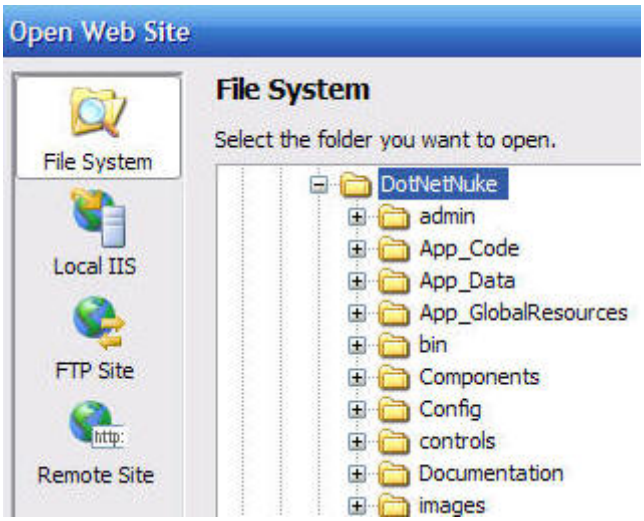
Download the *Install Version*. The source is bulky and not useful for module development because you don't want to develop a module on an installation that is not standard.

Create a directory on your hard drive. Do not put it under the *wwwroot* directory because it will most likely have permissions set that will make the installation difficult.

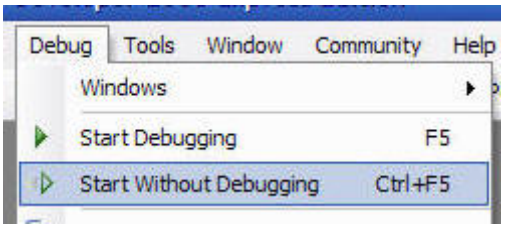
Open Visual Studio and select **File** then *Open Web Site*



Open the root of the web site



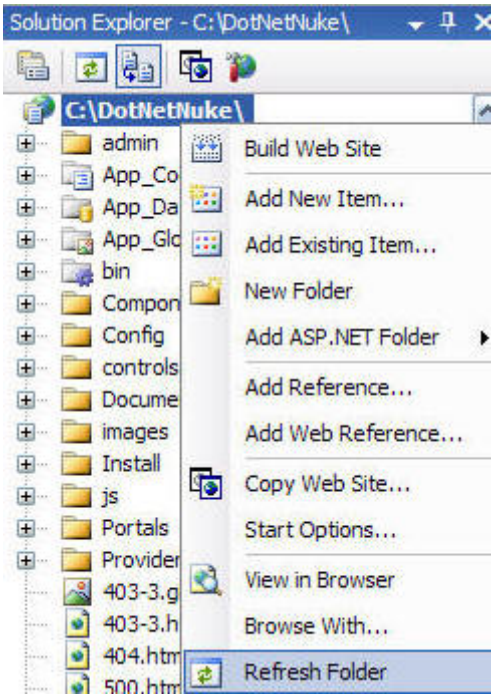
From the toolbar select **Debug** then ***Start Without Debugging***



The site will build and the internal web server will start

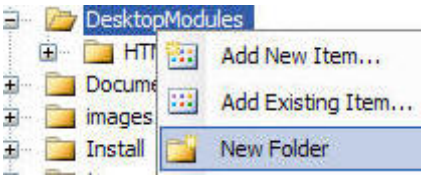
The site sets up using the default settings that are set to use SQL Server Express

Return to Visual Web Developer and *right-click* on the root of the website in the **Solution Explorer** and select ***Refresh Folder***

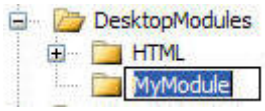


This will show the newly created folders such as the **DesktopModules** folder.

You can now *right-click* on it and create a New Folder

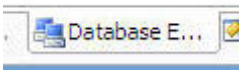


You can now start building your module in the directory.

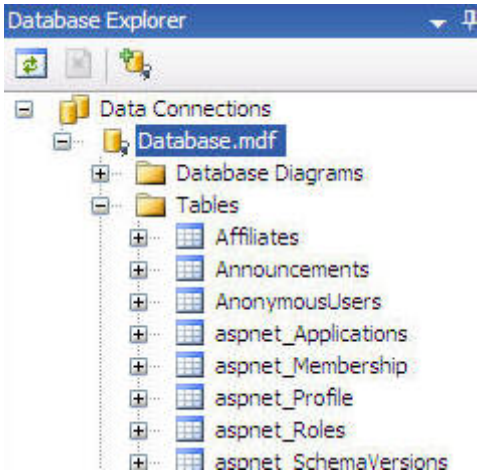


Visual Web Developer Express can cause SQL Server Express to lock.

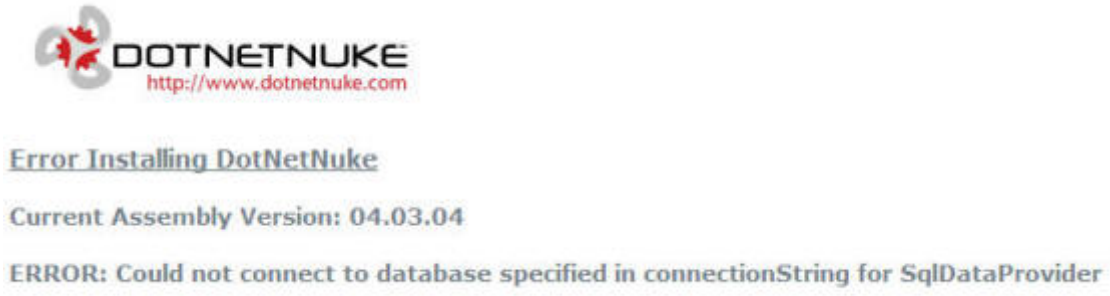
If you click on the Database Explorer tab...



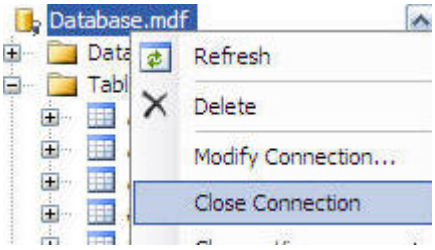
It will allow you to manage your SQL Server Express database...



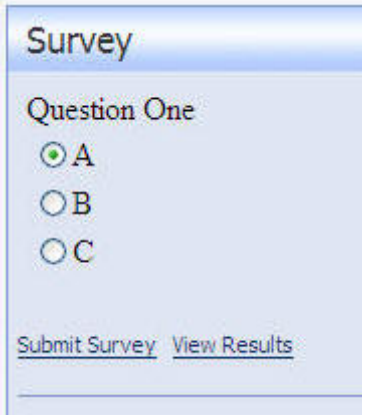
But then you get this error when you try to view the site in your web browser:



The problem is Visual Web Developer Express has locked the database. Right-click on the database and select *Close Connection*



Creating the Survey Module



This tutorial will show you how to create the 04.00.00 version of the DotNetNuke Core Survey Module.

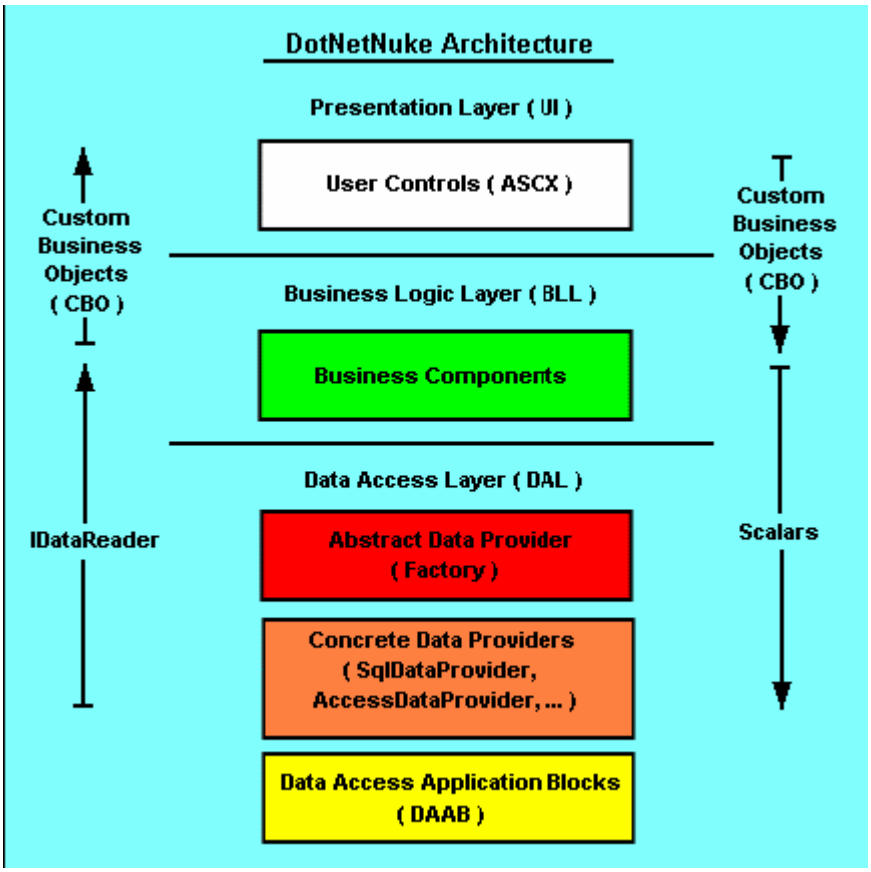
You can download C# Version of the Survey Module and the VB Version of the Survey Module from DotNetNuke.com

Are you ready to create the Module?

- You must have a DotNetNuke 4 website up and running.
- This tutorial will use SQL Server Express

The DotNetNuke Architecture

The DotNetNuke Architecture is traditionally represented using the following graphic:



Presentation Layer (UI) - This is the "face" of your module. This is where you put your buttons that people click on and the boxes they enter information in to.

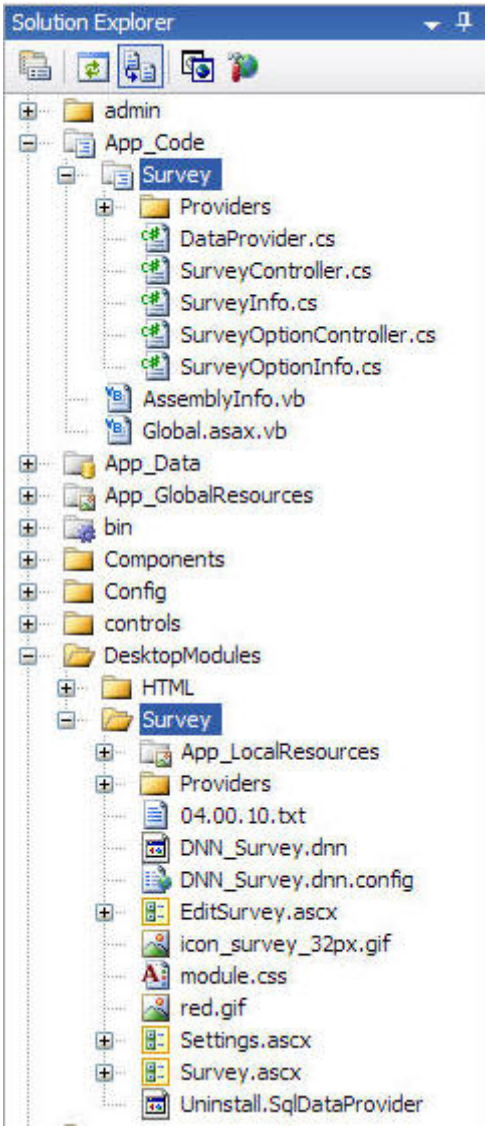
Business Logic Layer (BLL) - This is where your code that determines what your module will do will go.

Data Access Layer (DAL) - This is the DotNetNuke Data Access Layer. It consists of the Abstract Data Provider 1 or more Concrete Providers and the optional Data Access Application Blocks.

A DotNetNuke module resides in 2 directories.

The Web User Controls and their associated code behind files reside in the Module's directory that is in **DesktopModules** directory.

All other code (Data Access Layer and Business Logic Layer code) resides in the Module's directory that is in **App_Code** directory.



Developing Modules in C#

You can develop DotNetNuke modules in VB.NET or C# (or any other .Net language). However, if you are developing in a language other than VB.NET you have to add an entry like this to the web.config:

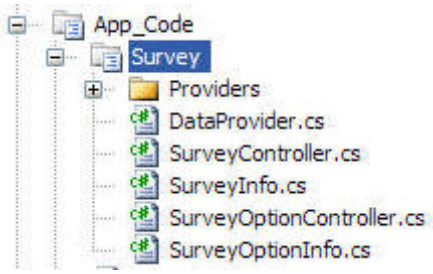
```
<codeSubDirectories>
  <add directoryName="Survey" />
</codeSubDirectories>
```

This node is in the <system.web><compilation> node, (there is a commented out example in web.config).

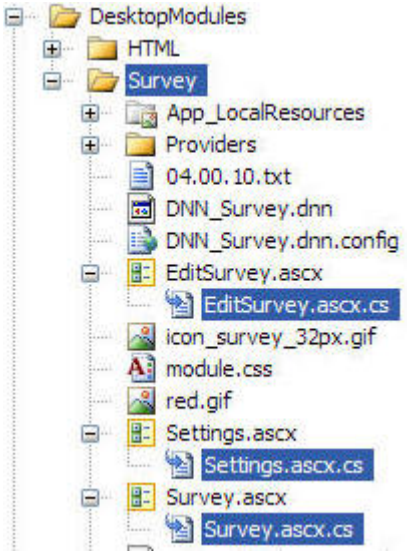
You must perform this step because the main project is written in VB.NET and you must instruct the compiler to compile the indicated directory in C#.

```
<!-- set debugmode to false for running application -->
<compilation debug="false" strict="false">
  <buildProviders>
    <remove extension=".resx" />
    <remove extension=".resources" />
  </buildProviders>
  <assemblies>
    <add assembly="Microsoft.VisualBasic, Version=8.0.0
    <add assembly="System.DirectoryServices, Version=2.
    <add assembly="System.Design, Version=2.0.0.0, Cult
    <add assembly="System.Management, Version=2.0.0.0,
  </assemblies>
  <!-- register your app_code subfolders to generate
  granular assemblies during compilation
  -->
  <codeSubDirectories>
      <add directoryName="Survey" />
  </codeSubDirectories>
  </compilation>
```

This entry is only needed for the files placed in the directory under the **App_Code** directory.



The code-behind files that are associated User Controls residing under the **DesktopModules** directory do not need an entry in the web.config file.



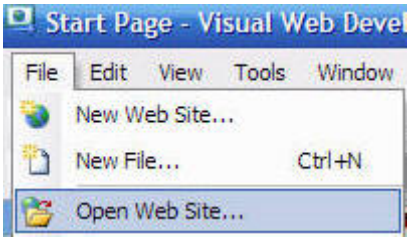
Creating the Module

We will create the module using the following steps:

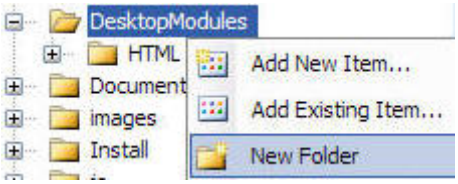
- Create the folders
- Create the files
- Create the content

Creating the folders under DesktopModules

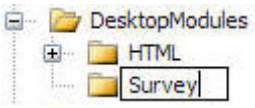
Open your DotNetNuke website in Visual Studio or Visual Web Developer Express by selecting **File** from the menu bar and selecting **Open Web Site**.



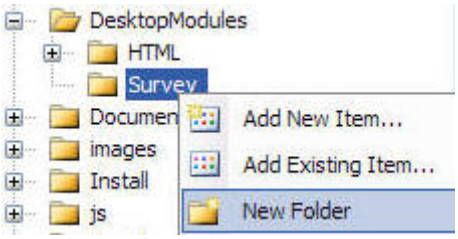
In the Solution Explorer, *right-click* on the **DesktopModules** folder and select **New Folder**.



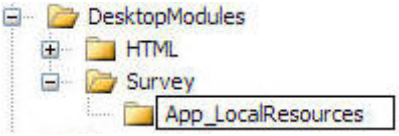
Name the folder **Survey**.



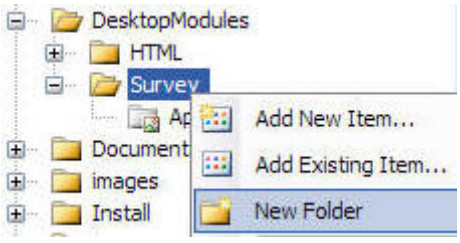
Right-click on the **Survey** folder and select **New Folder**.



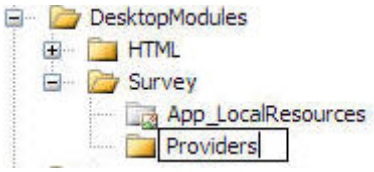
Name the folder **App_LocalResources**. This is where *resource* files will reside. These resource files will be used to provide *Localization*. Localization will allow the module to be used in different languages by simply replacing the text in the resource files.



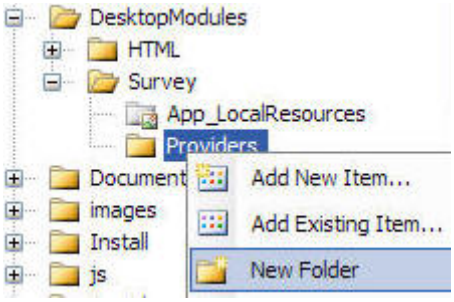
Right-click on the **Survey** folder again and select **New Folder**.



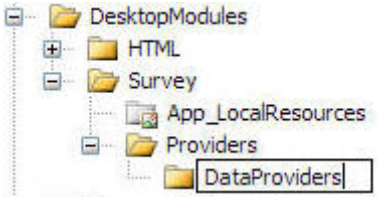
Name the folder **Providers**.



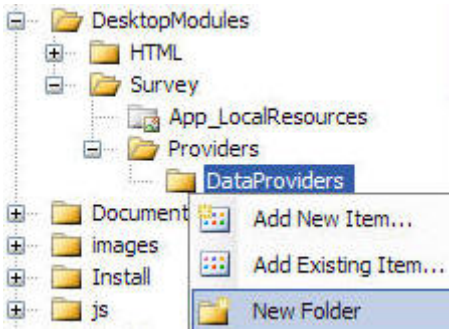
Right-click on the **Providers** folder and select **New Folder**.



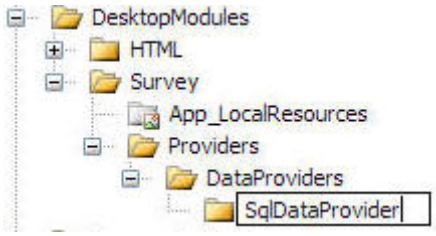
Name the folder **DataProviders**. This is where the database installation scripts for the module will reside.



Right-click on the **DataProviders** folder and select **New Folder**.

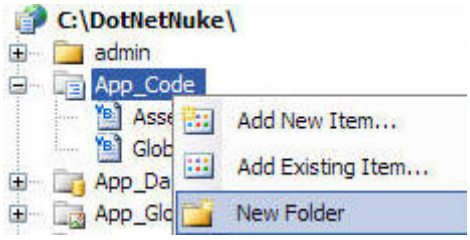


Name the folder **SQLDataProvider**. We will only create Microsoft SQL setup scripts in this tutorial. However, if we wanted to create data providers and set-up scripts for other databases such as Oracle, MySQL, or Firebird, we would create a folder for each of these.



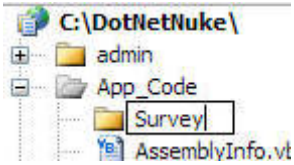
Creating the folders under App_Code

Right-click on the **App_Code** folder and select **New Folder**.

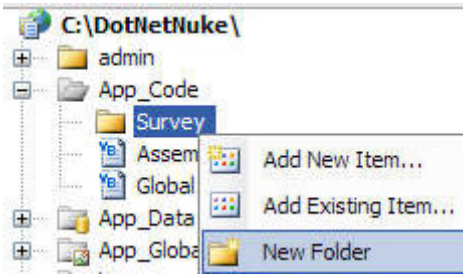


Name the folder **Survey**. This is where the files that are pure class files (that are not associated with a User Control) will reside.

The code files that will be placed here will comprise the **BLL** and **DAL** layer.



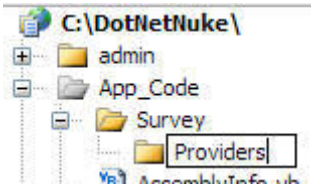
Right-click on the **Survey** folder and select **New Folder**.



Name the folder **Providers**. This is where the *concrete provider* class file will reside. The concrete provider is the class that actually connects to a specific database. The class that connects to the concrete class is an abstract class.

In this tutorial, only a Microsoft SQL provider class will be created.

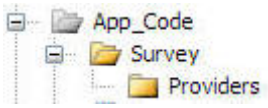
However, if we wanted to create data providers for other databases such as Oracle, MySQL, or Firebird, we would also place those *concrete providers* in this directory.

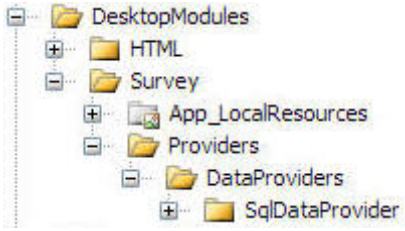


What Did We Just Do?

We have created the folder structure for the Survey Module.

We have placed directories under the **DesktopModules** folder and under the **App_Code** folder.





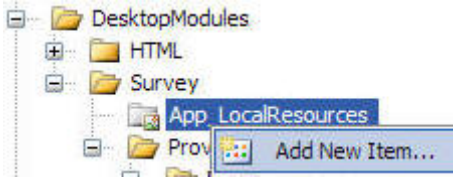
Creating the files under DesktopModules

We will create the files needed for the module first, and add the content in a later step.

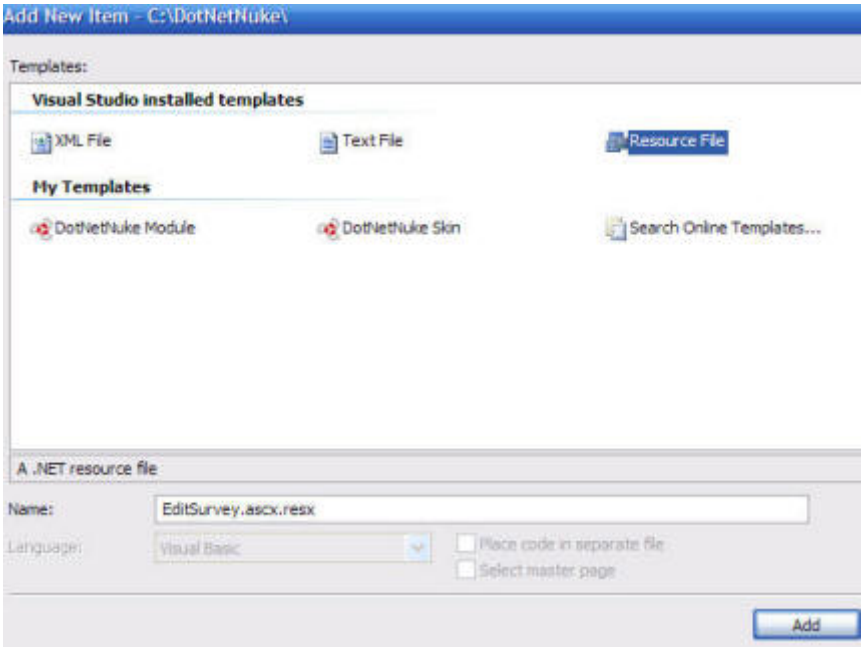
Creating the resource files that will be used for Localization.

We will first create the Localization files that will be used to provide the text for the user controls of the module. This will allow a portal administrator to replace these files and display the text in a different language.

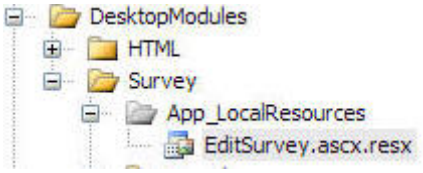
Right-click on the **App_LocalResources** directory (that is under the **Survey** directory that is under the **DesktopModules** folder) and select *Add New Item*.



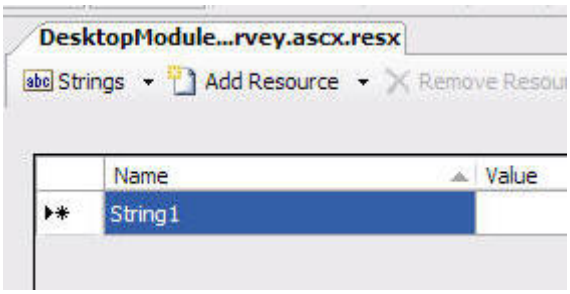
When the **Add New Item** menu comes up, select **Resource File** and enter *EditSurvey.ascx.resx* in the **Name** box and click the **Add** button.



The *EditSurvey.ascx.resx* will appear in the Solution Explorer.

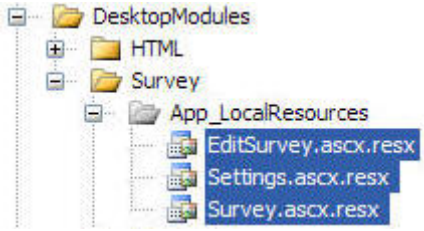


The file will also open in the **Edit** window. Close this for now.



Repeat the steps used to create the *EditSurvey.ascx.resx* file to also create:

- Settings.ascx.resx
- Survey.ascx.resx



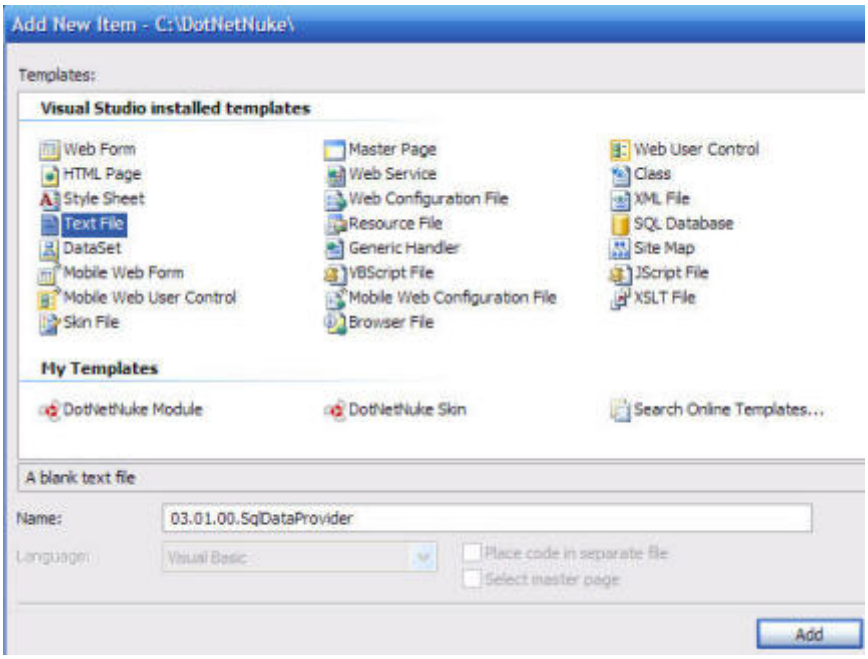
Creating the SQL script files

We will now create the SQL script files. We are placing these files in this location to allow the DotNetNuke framework to automatically place them in the installable module package when you package your DNN Module (this is covered in another chapter in this guide).

Right-click on the **SqlDataProvider** directory (that is under the **DataProvider** directory that is under the **Providers** directory that is under the **Survey** directory that is under the **DesktopModules** folder) and select *Add New Item*.

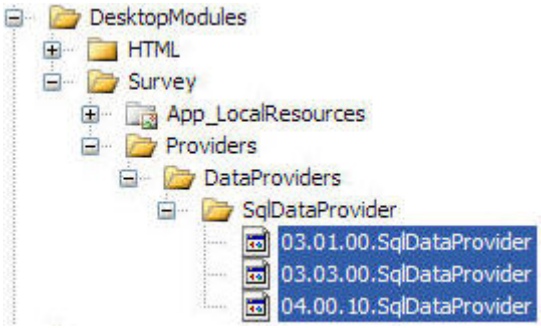
When the **Add New Item** menu comes up, select **Text File** and enter *03.01.00.SqlDataProvider* in the **Name** box and click the **Add** button.

*(The file will also open in the **Edit** window. Close this for now.)*



Repeat the steps used to create the *03.01.00.SqlDataProvider* file to also create:

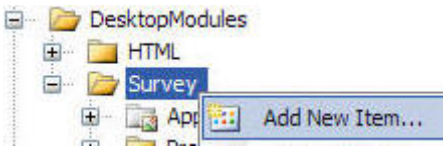
- 03.03.00.SqlDataProvider
- 04.00.10.SqlDataProvider



Creating the Text Files

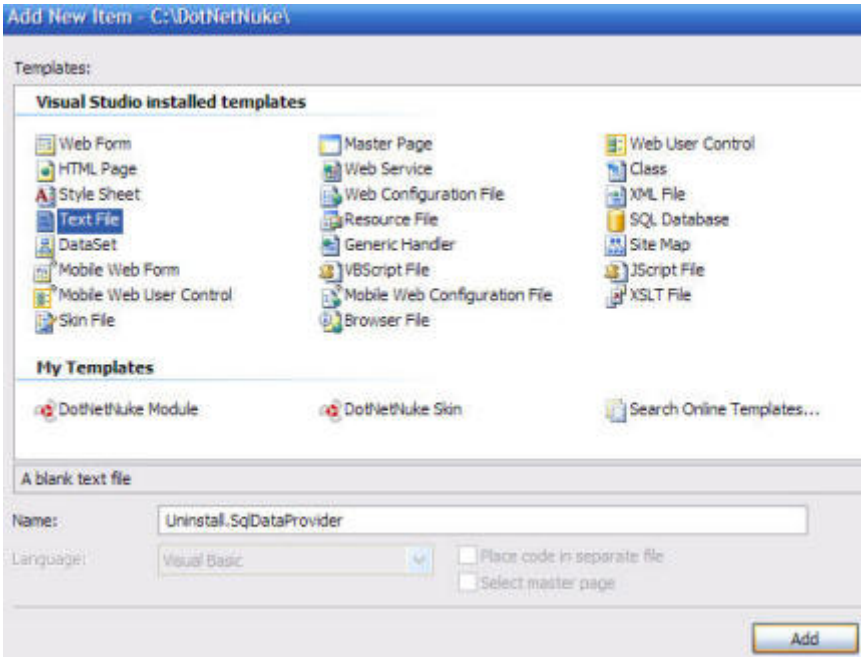
We will now create the various text files that will be needed for the module and that reside in the **DesktopModules** folder.

Right-click on the **Survey** directory (that is under the **DesktopModules** folder) and select *Add New Item*.



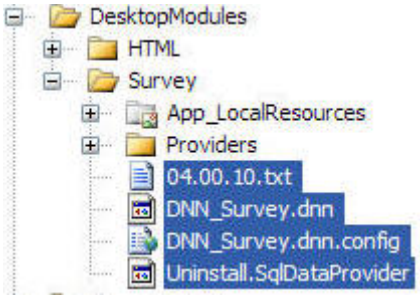
When the **Add New Item** menu comes up, select **Text File** and enter *Uninstall.SqlDataProvider* in the **Name** box and click the **Add** button.

*(The file will also open in the **Edit** window. Close this for now.)*



Repeat the steps used to create the *Uninstall.SqlDataProvider* file to also create:

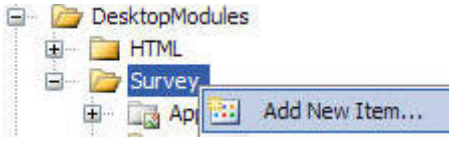
- 04.00.10.txt
- DNN_Survey.dnn
- DNN_Survey.dnn.config

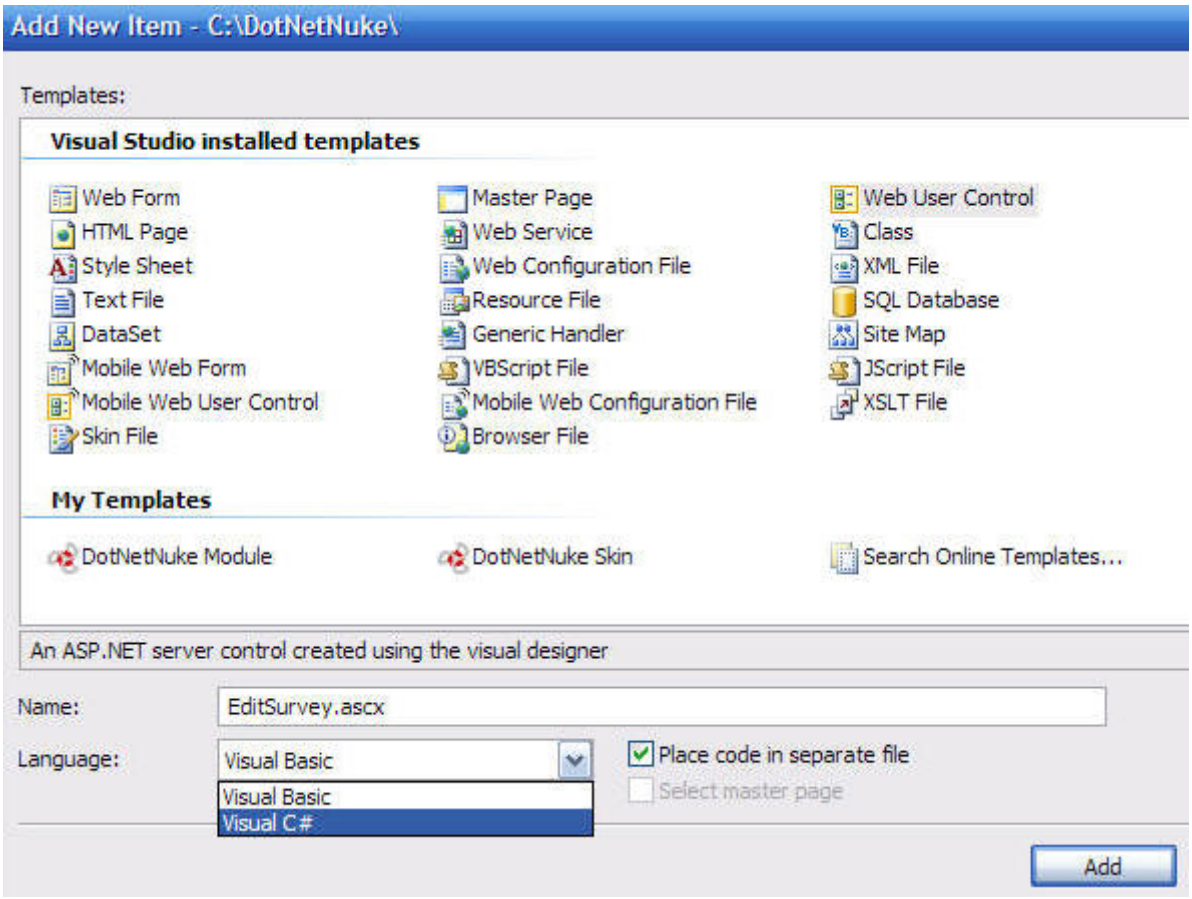


Creating the User Controls

We will now create the User Control files that will be needed for the module and that reside in the **DesktopModules** folder.

Right-click on the **Survey** directory (that is under the **DesktopModules** folder) and select *Add New Item*.





When the **Add New Item** menu comes up:

- Select Web User Control
- Enter *EditSurvey.ascx* in the Name: box.
- Select a language in the Language drop-down
- Click the box next to Place code in a separate file
- Click the Add button.

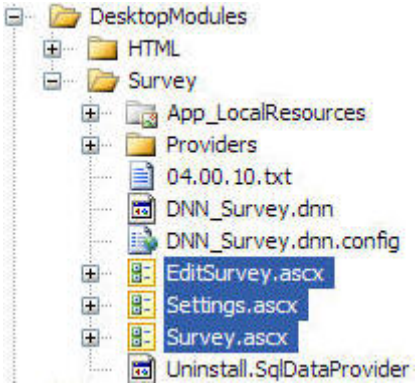
*(The file will also open in the **Edit** window. Close this for now.)*

(a code behind file for the user control will also be created. Either EditSurvey.cs or EditSurvey.vb depending on the language you selected.)

(Visual Studio will show errors for the file. Ignore these for now as they will be fixed when content is inserted for the files in a later step.)

Repeat the steps used to create the *EditSurvey.ascx* file to also create:

- Settings.ascx
- Survey.ascx

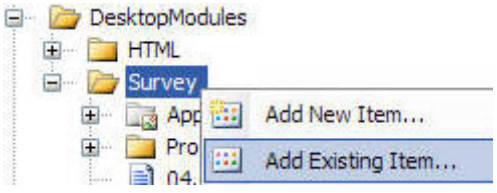


Creating the Remaining Files for DesktopModules folder

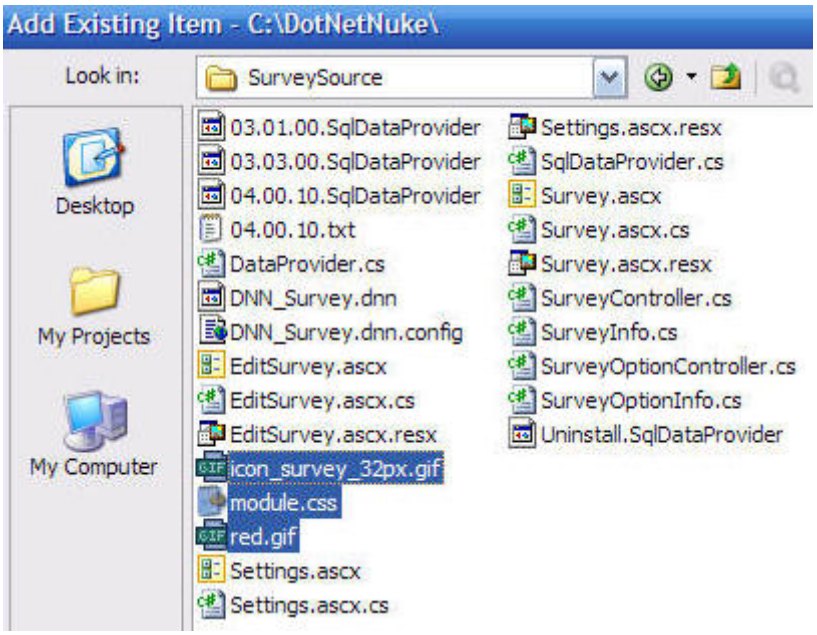
You can obtain the remaining files from the *Survey Module* source package that can be downloaded DotNetNuke.com

- icon_survey_32px.gif
- red.gif
- module.css

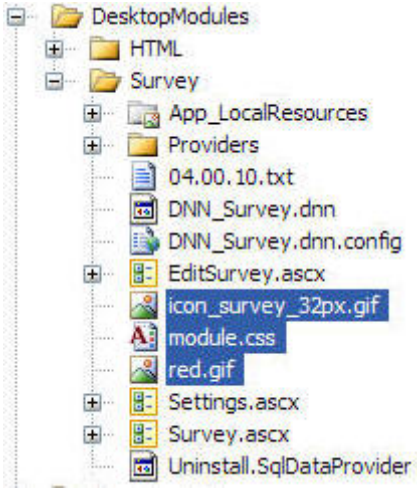
To import them, unzip them from the *.zip* file they are contained in and *right-click* on the **Survey** directory in Visual Studio (that is under the **DesktopModules** folder) and select *Add Existing Item*.



Navigate to the directory that you placed the files in and select the files by holding down the **Ctrl** key while you click on each one. Click the **Add** button



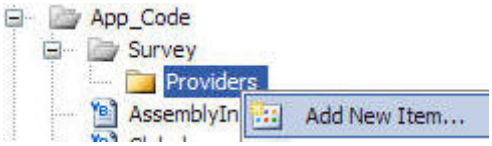
The files will be imported.



Creating the files under App_Code Directory

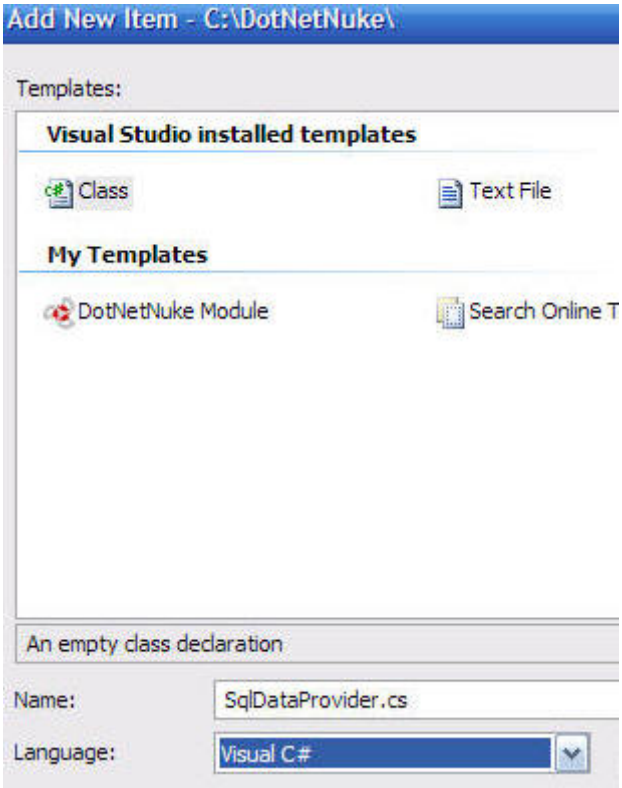
We will now create the files that will be pure class files. These files will reside in the **App_Code** directory.

Right-click on the **Providers** directory (that is under the **Survey** directory that is under the **App_Code** folder) and select *Add New Item*.

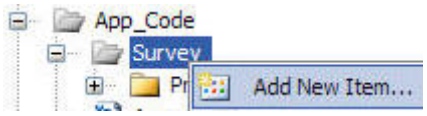


When the **Add New Item** menu comes up, select **Class** and enter *SqlDataProvider.cs* if you are planning to code in C# (name the file *SqlDataProvider.vb* if you are planning to code in VB*)* in the **Name** box, select a language in the **Language** drop-down, and click the **Add** button.

*(The file will also open in the **Edit** window. Close this for now.)*

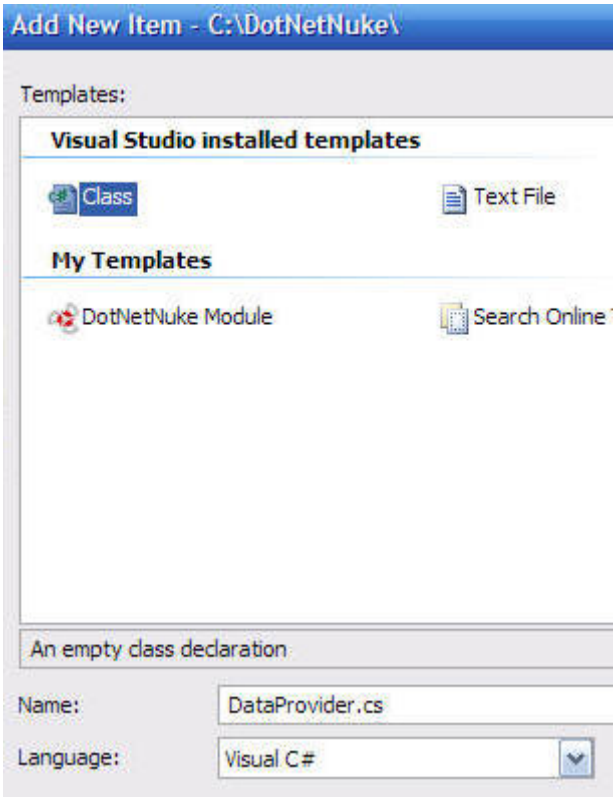


Right-click on the **Survey** directory (that is under the **App_Code** folder) and select *Add New Item*.



When the **Add New Item** menu comes up, select **Class** and enter *DataProvider.cs* if you are planning to code in C# (name the file *DataProvider.vb* if you are planning to code in VB*)* in the **Name** box, select a language in the **Language** drop-down, and click the **Add** button.

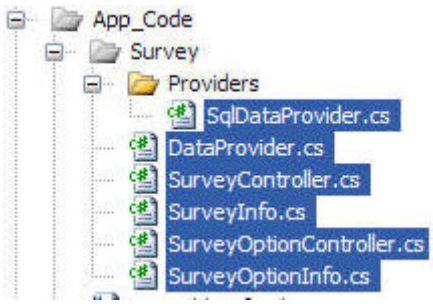
*(The file will also open in the **Edit** window. Close this for now.)*



Repeat the steps above to also create (if you are planning to code in C#):

- SurveyController.cs
- SurveyInfo.cs
- SurveyOptionController.cs
- SurveyOptionInfo.cs

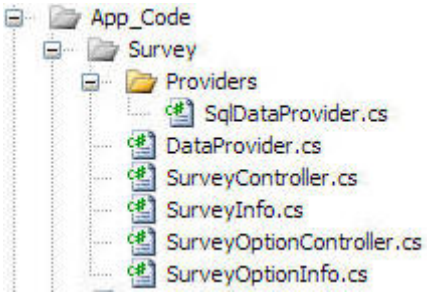
(if you are planning to code in VB, use the *.vb* extension for the files instead of *.cs*)

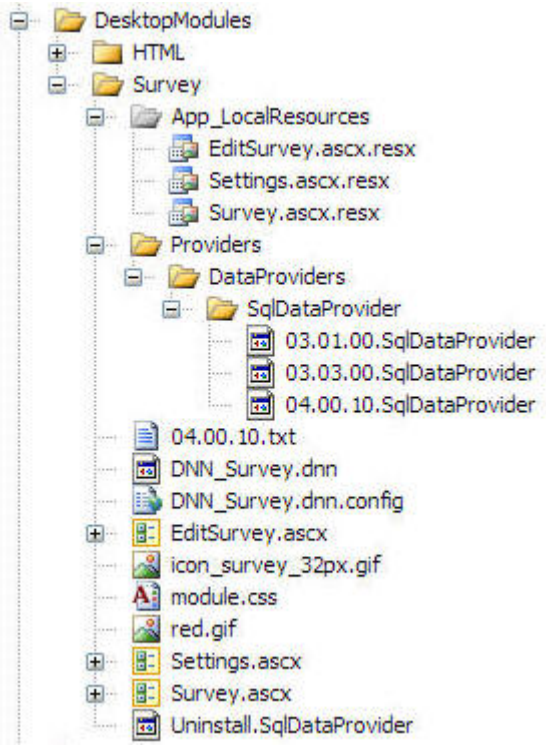


What Did We Just Do?

We have created all the files for the Survey Module.

We have placed them in directories under the **App_Code** folder and under the **DesktopModules** folder.





Create the Content

Now that the files have been created, you will now insert the content for each file. You can obtain this content by downloading the *Survey Module* from DotNetNuke.com.

The source code for the Survey module is listed in DotNetNuke 4.0 Module Developers Guide (Part 2) available at DotNetNuke.com *(**.cs** is for the C# version of the file and **.vb** for the Visual Basic version of the file)*:

App_Code (folder)

- DataProvider[.cs / .vb]
- SurveyController[.cs / .vb]
- SurveyInfo[.cs / .vb]
- SurveyOptionController[.cs / .vb]
- SurveyOptionInfo[.cs / .vb]

Providers (folder)

- SqlDataProvider[.cs / .vb]

DesktopModules (folder)

- 04.00.10.txt
- DNN_Survey.dnn [.cs / .vb]
- DNN_Survey.dnn.config [.cs / .vb]
- EditSurvey.ascx [.cs / .vb]
- EditSurvey.ascx[.cs / .vb]
- icon_survey_32px.gif
- module.css
- red.gif
- Settings.ascx [.cs / .vb]
- Settings.ascx[.cs / .vb]
- Survey.ascx [.cs / .vb]
- Survey.ascx[.cs / .vb]
- Uninstall.SqlDataProvider

App_LocalResources (folder)

- EditSurvey.ascx.resx
- Settings.ascx.resx
- Survey.ascx.resx

Providers (folder)

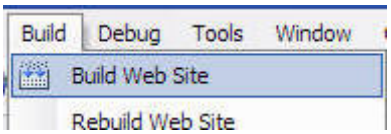
DataProviders (folder)

SqlDataProvider (folder)

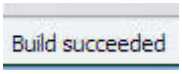
- 03.01.00.SqlDataProvider
- 03.03.00.SqlDataProvider
- 04.00.10.SqlDataProvider

Build the Site

After the content for each file has been inserted, you will then build the site. From the menu bar, select **Build** then **Build Web Site**

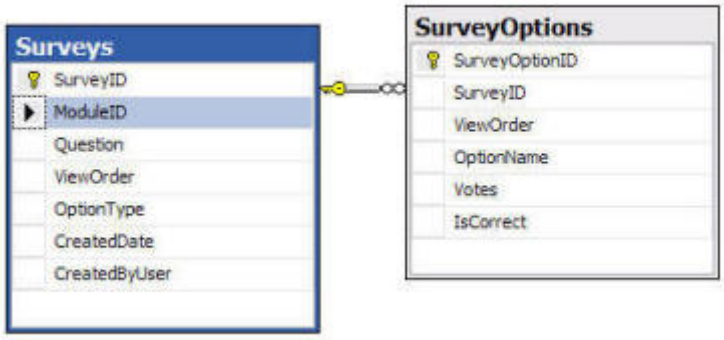


The site should build without errors. If it has errors, download the code from DotNetNuke.com above and compare the code to your code.



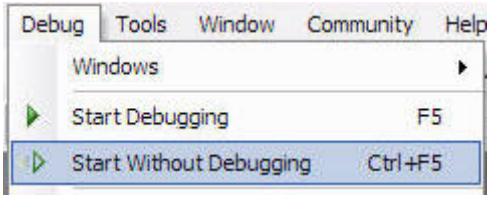
Create the Tables and Stored Procedures

The code is complete but the module will not work until you create the tables and stored procedures and set up the module configuration.



Open the DotNetNuke Site

From the menu bar in Visual Studio, select **Debug**, then *Start Without Debugging*



The site will come up

Run the Database Scripts

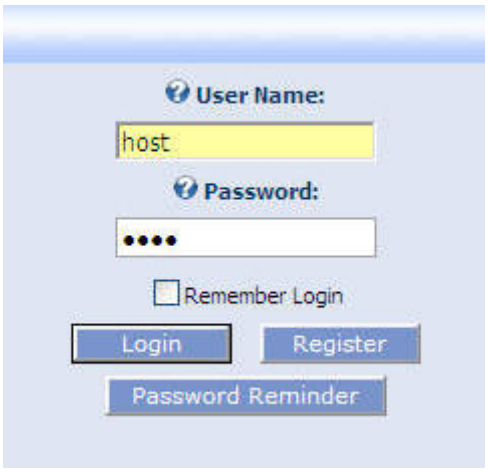
You will now execute the three database scripts needed to create the tables and stored procedures the Module needs.

The reason there are three scripts is that each script is from a different version of the Module. When the module is installed, the DotNetNuke framework will run each SQL script in the Module package in order.

This is the mechanism that allows you to upgrade a module. For example, if the portal already has the Module version 03.01.00 installed it will not run that SQL script and will only run the SQL scripts with a higher number.

Click **LOGIN**

Log in as "**host**". The password (if you haven't already changed it) is also "**dnnhost**"

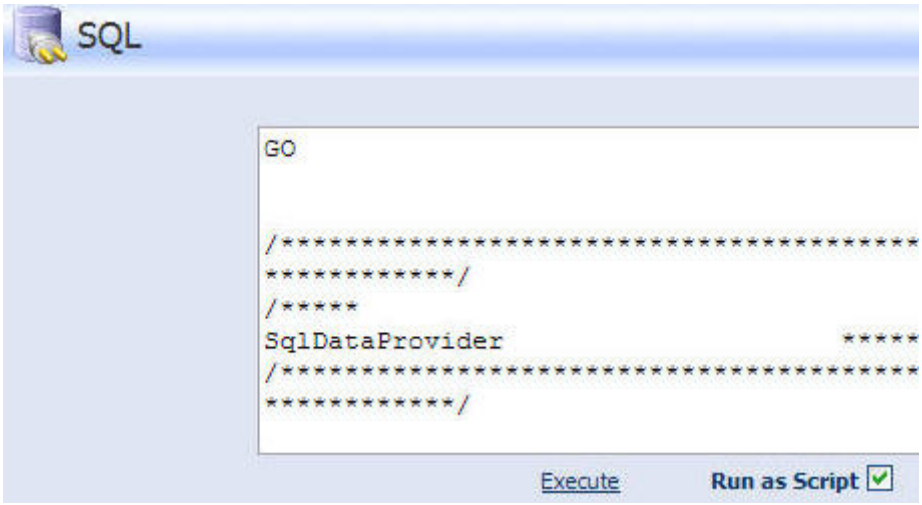


Click on the **HOST** menu and select **SQL**

Obtain the contents for the 03.01.00.SqlDataProvider from DotNetNuke.com (or from the DotNetNuke 4.0 Module Developers Guide (Part 2) available at DotNetNuke.com) and insert the entire SQL Script into the SQL box in your DotNetNuke website.

```
/************************************************************/
/*****              SqlDataProvider                     *****/
/*****                                                  *****/
/*****                                                  *****/
/***** Note: To manually execute this script you must   *****/
/*****       perform a search and replace operation     *****/
/*****       for {databaseOwner} and {objectQualifier}  *****/
/*****                                                  *****/
/************************************************************/

if exists (select * from dbo.sysobjects where id = object_id(N'
drop procedure {databaseOwner}{objectQualifier}GetSurveys
GO

if exists (select * from dbo.sysobjects where id = object_id(N'
drop procedure {databaseOwner}{objectQualifier}GetSurvey
GO

if exists (select * from dbo.sysobjects where id = object_id(N'
drop procedure {databaseOwner}{objectQualifier}AddSurvey
```

Select the *Run as Script* box and click **Execute**.



Watch the progress bar in your web browser because when the script has been run the text in the window will shift and then... nothing...

Next, select everything in the SQL box and delete it.

Paste the contents of the following scripts in the box, one at a time, in order:

- 03.03.00.SqlDataProvider

- 04.00.10.SqlDataProvider

Make sure that you Select the Run as Script box and click Execute each time and that you clear the contents of the SQL box each time.

If you make a mistake or receive an error, run the following script and re-run each script in order:

Uninstall.SqlDataProvider

What Did We Just Do?

We ran a SQL script that created the tables and the stored procedures.

You will notice that the script is written like this:

```
CREATE TABLE {databaseOwner}{objectQualifier}Surveys
```

rather than the normal:

```
CREATE TABLE [dbo][Surveys]
```

The script commands "**{databaseOwner}**" and "**{objectQualifier}**" indicate that they are to be replaced by configuration settings in the **web.config** file. Normally "**{databaseOwner}**" is set to ".dbo" and "**{objectQualifier}**" is set to nothing (it would not have a setting). However, if alternate settings were indicated in the **web.config** file, those settings would be inserted into the script.

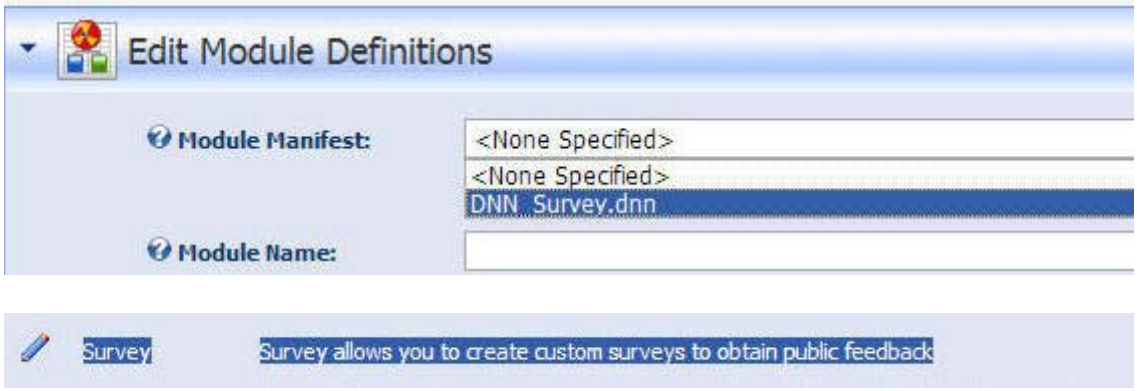
You must have the **Run as Script** box checked for this replacement to happen.

Configure the Module

From the **Host** menu in your DotNetNuke website, select **Module Definitions**

From the **Module Definitions** menu, select **Create New Module**

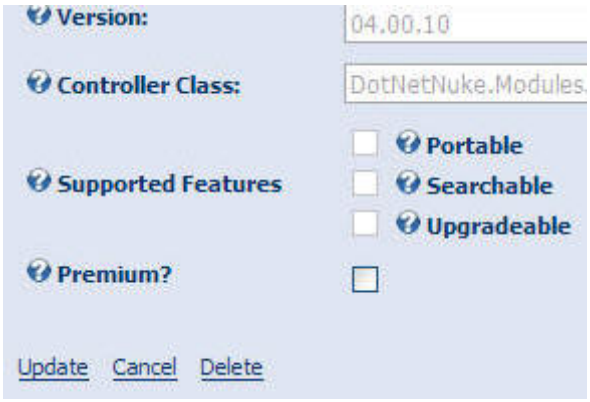
The **DNN_Survey.dnn** file (created in the earlier step) will be detected. Select it from the *drop-down* and click the **Install** link.



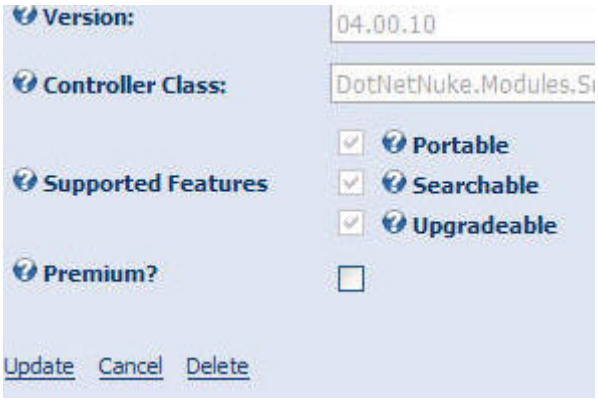
The Survey Module definition will now show in the Module definitions list. Click the **Edit** link next to the Survey Module definition.

The optional interfaces (Portable, Searchable, Upgradeable) will not be checked.

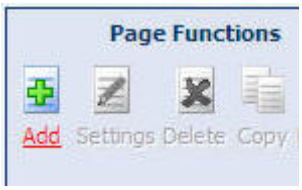
Click the **Update** Link



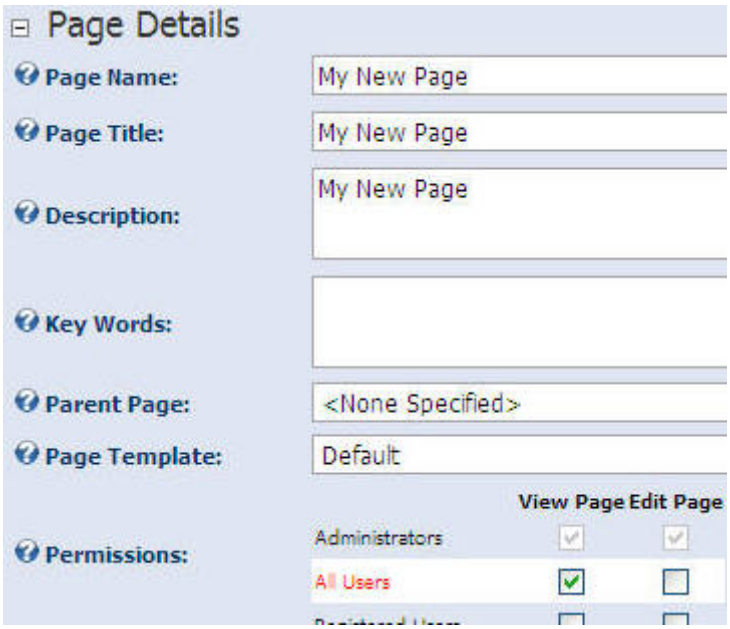
The Module Configuration will now be properly updated



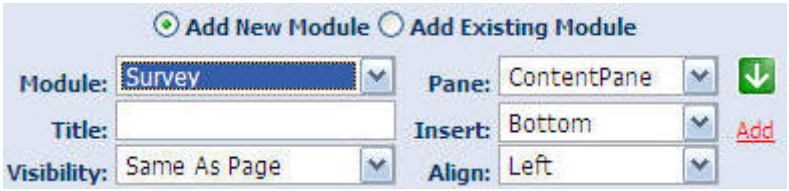
Next, create a page to place the module on. On the Administrator control panel, click the *Add* link under the *Page Functions* section.



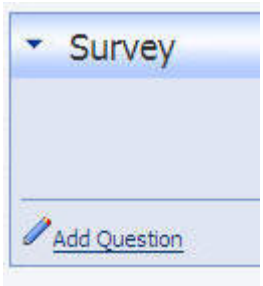
Enter details for the page, ensuring that **All Users** are selected under the *View Page* column. When you have entered the information, click the **Update** link.



Next, from the Administrator control panel, select *Survey* from the **Module** drop-down list and click the *Add* link.



The Survey Module will now appear.



Connect Your Module to the Database

DAL & DAL+

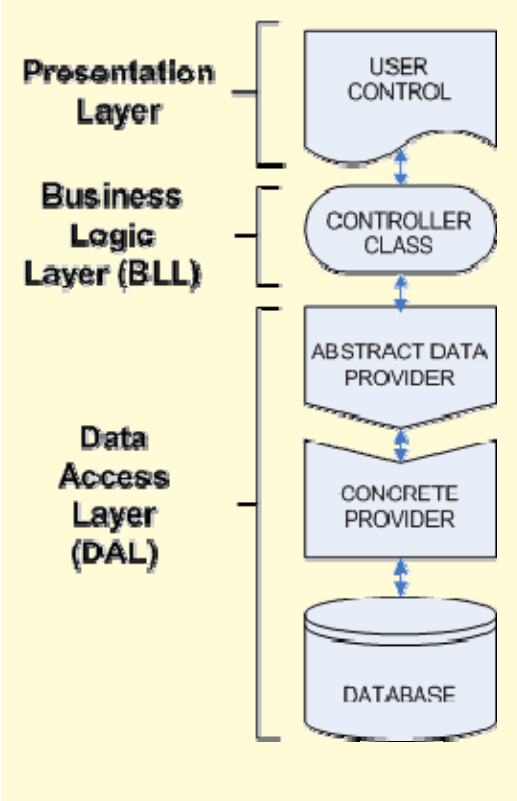
DotNetNuke and Data Access

When you create a DotNetNuke module that needs to connect to the database, you have the option of coding the data access methods manually or leveraging the DotNetNuke framework.

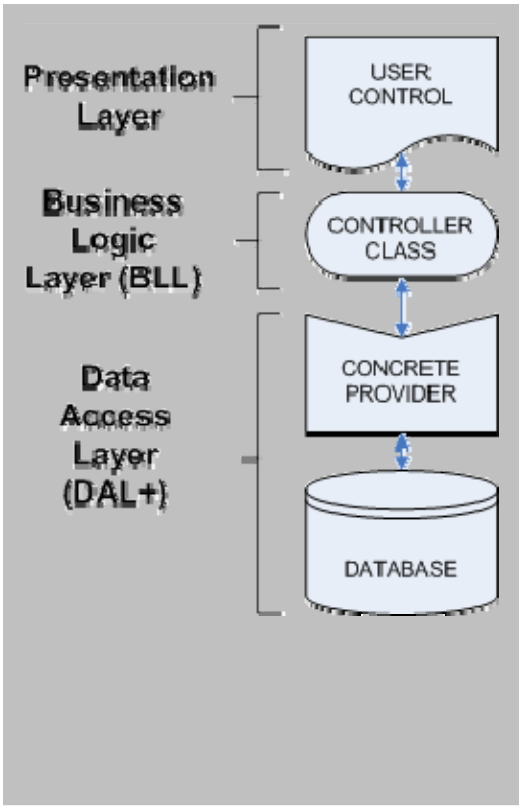
The DotNetNuke framework provides a full featured Data Access Layer (DAL). The DAL also includes a special subset of methods commonly refereed to as the DAL+

The DAL and the DAL+

The following diagram shows an overview of the DAL and the DAL+ :



The DAL



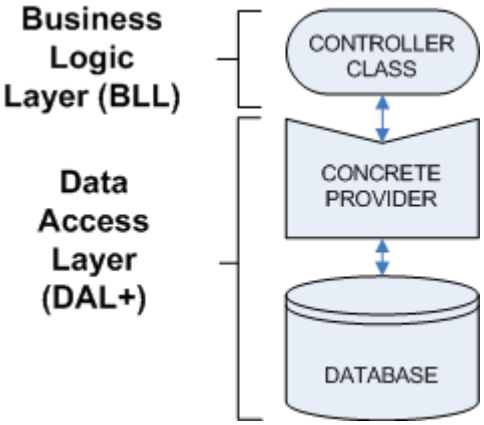
The DAL+

The Purpose of the DAL (and the DAL+)

The DAL has this purpose:

To allow DotNetNuke (and it's modules) to communicate with any data source.

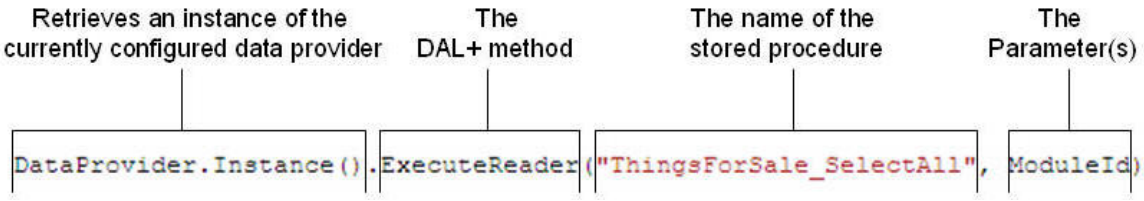
A Close-up Look at the DAL+



The DAL+ does not require an **Abstract Data Provider** and it does not require that you code a custom **Concrete Provider**. The DAL+ is a subset of the DAL. It is comprised of 4 methods that exist in the DotNetNuke framework's currently configured **Concrete Provider**. The methods are:

- **ExecuteNonQuery** - Used to execute a stored procedure that will not return a value.
- **ExecuteReader** - Used to execute a stored procedure that will return multiple records.
- **ExecuteScalar** - Used to execute a stored procedure that will return a single value.
- **ExecuteSQL** - Used to execute a sql statement.

Below is an explanation of the format used to implement the DAL+ using the **ExecuteReader** method:



The DAL+ allows you to use code such as this (in the **Controller Class**) to connect to the currently configured database:

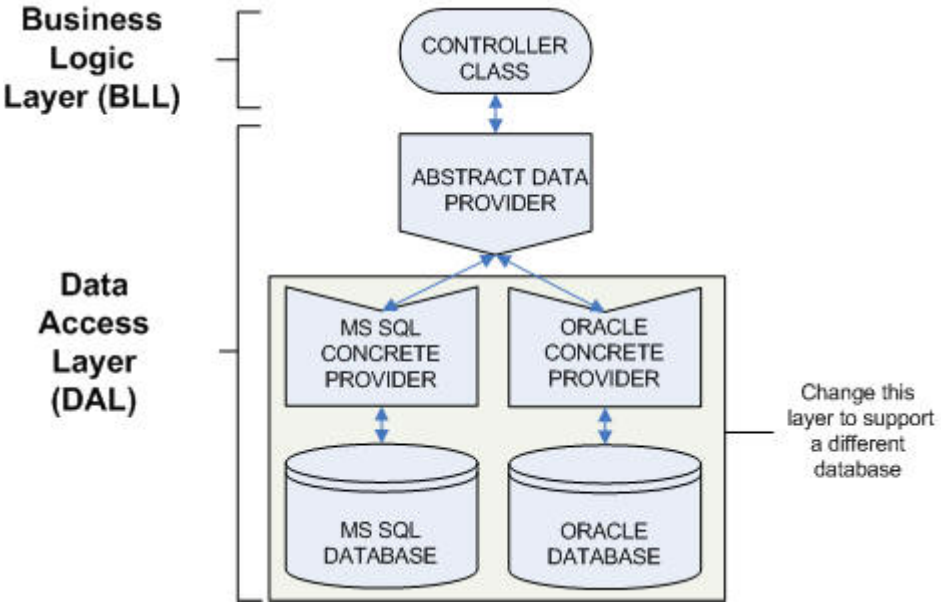
```
Public Shared Function ThingsForSale_SelectAll(ByVal ModuleId As Integer) As
List(Of ThingsForSaleInfo)
    Return CBO.FillCollection(Of
ThingsForSaleInfo)(CType(DataProvider.Instance().ExecuteReader("ThingsForSale_Sel
ectAll", ModuleId), IDataReader))
End Function
```

(note: CBO.FillCollection is a special method provided by the DotNetNuke framework to hydrate a custom collection. See further reading at the end of this article for more information)

However, unlike the DAL, the DAL+ is not *100% portable* to other data sources. For example, if a module is developed using the DAL+ that retrieves data using one stored procedure, an alternate database must be able to perform the exact same functionality using only one stored procedure. In some cases this is not possible due to differences in databases.

However, in those instances where you will not need to run your module on alternate databases (for example for internal development) it is recommended that you use the DAL+. The code savings is significant.

A Close-up Look at the DAL



The DAL does require that you code an **Abstract Provider** and one or more custom **Concrete Providers**. This allows you to create modules that are *100% portable* to other databases.

The **Abstract Provider** reads the settings in the *web.config* file to determine what the currently configured database is and exposes data access methods. In the example below the method is **GetSurveys** which is declared *MustOverride*.

```
' return the provider
Public Shared Shadows Function Instance() As DataProvider
  Return objProvider
End Function

' dynamically create provider
Private Shared Sub CreateProvider()
  objProvider = CType(Framework.Reflection.CreateObject("data",
"DotNetNuke.Modules.Survey", ""), DataProvider)
End Sub

' methods to be overridden by the concrete provider
Public MustOverride Function GetSurveys(ByVal ModuleId As Integer) As IDataReader
```

The **Concrete Provider** overrides the methods in the **Abstract Provider** and performs the data access task.

```
Public Class SqlDataProvider
  Inherits DataProvider

Public Overrides Function GetSurveys(ByVal ModuleId As Integer) As IDataReader
  Return CType(SqlHelper.ExecuteReader(ConnectionString, DatabaseOwner &
ObjectQualifier & "GetSurveys", ModuleId), IDataReader)
End Function
```

(note: SqlHelper is a special method provided by the DotNetNuke framework (using the Microsoft Application Blocks) to reduce the code needed for data access. See further reading at the end of this article for more information)

The Controller Class uses *DataProvider.Instance()* to call methods in the **Abstract Provider** which calls the corresponding overridden methods in the **Concrete Provider**.

```
Public Shared Function GetSurveys(ByVal ModuleId As Integer) As List(Of SurveyInfo)
Dim SurveyInfolist As List(Of SurveyInfo) = New List(Of SurveyInfo)
 Using dr As IDataReader = DataProvider.Instance().GetSurveys(ModuleId)
  While dr.Read
   Dim SurveyInfo As SurveyInfo = New SurveyInfo
   SurveyInfo.SurveyId = Convert.ToInt32(dr("SurveyId"))
   SurveyInfo.Question = Convert.ToString(dr("Question"))
   SurveyInfo.OptionType = Convert.ToString(dr("OptionType"))
   SurveyInfo.ViewOrder = Convert.ToInt32(ConvertNullInteger(dr("ViewOrder")))
   SurveyInfo.CreatedByUser = Convert.ToInt32(dr("CreatedByUser"))
   SurveyInfo.CreatedDate = Convert.ToDateTime(dr("CreatedDate"))
   SurveyInfolist.Add(SurveyInfo)
  End While
 End Using
Return SurveyInfolist
End Function
```

Leverage the DotNetNuke Framework

Using the DAL+ allows a module developer to write less code to access the database than they would if they did not use the DotNetNuke framework. The DAL+ has its limitations and the DAL is provided in those cases where 100% portability is required.

Converting the Survey module to use the DAL+

The DAL vs. the DAL+

Using the normal DotNetNuke Data Access Layer (DAL) design, we would create a database provider class (Concrete Provider) that communicated with the database and overrode methods in an abstract class (Abstract Data Provider). The abstract class sits between the concrete provider class and the Business Logic Layer (Controller Class). This would have allowed us to substitute an alternate database provider class to communicate with other databases.

The DAL+ does not require an Abstract Data Provider and it does not require that you code a custom Concrete Provider. The DAL+ is a subset of the DAL. It is comprised of 4 methods:

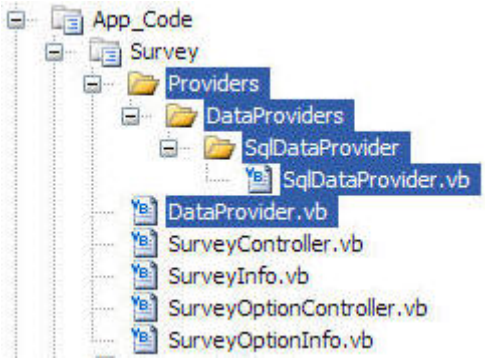
- **ExecuteNonQuery** - Used to execute a stored procedure that will not return a value.
- **ExecuteReader** - Used to execute a stored procedure that will return multiple records.
- **ExecuteScalar** - Used to execute a stored procedure that will return a single value.
- **ExecuteSQL** - Used to execute a sql statement.

However, unlike the DAL, the DAL+ is not 100% portable to other data sources. For example, if a module is developed using the DAL+ that retrieves data using one stored procedure, an alternate database must be able to perform the exact same functionality using only one stored procedure. In some cases this is not possible due to differences in databases.

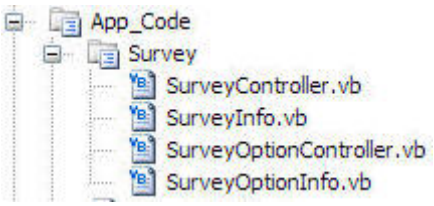
Converting the Survey Module to use the DAL+

As a demonstration of the DAL+, the Survey module (that uses the traditional DAL) will be converted to use the DAL+ (you can download the original C# Version of the Survey Module and the VB Version of the Survey Module from the Survey Project page on DotNetNuke.com).

To replace the DAL layer, we only need to alter the files that reside in the **App_Code** directory.



Delete the highlighted files above so that only the following files remain:



Next, open the **SurveyController.vb** and **SurveyOptionController.vb** (or **SurveyController.cs** and **SurveyOptionController.cs** if you're using the C# version) and replace the **GetSurveys**, **GetSurvey**, **DeleteSurvey**, **AddSurvey**, **UpdateSurvey**, **GetSurveyOptions**, **DeleteSurveyOption**, **AddSurveyOption**, **UpdateSurveyOption**, and **AddSurveyResult** methods with the following code:

VB.NET

SurveyController.vb:

```
Public Shared Function GetSurveys(ByVal ModuleId As Integer) As List(Of
SurveyInfo)
    Dim SurveyInfolist As List(Of SurveyInfo) = New List(Of SurveyInfo)
    Using dr As IDataReader =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurveys", ModuleId)
    While dr.Read
        Dim SurveyInfo As SurveyInfo = New SurveyInfo
        SurveyInfo.SurveyId = Convert.ToInt32(dr("SurveyId"))
        SurveyInfo.Question = Convert.ToString(dr("Question"))
        SurveyInfo.OptionType = Convert.ToString(dr("OptionType"))
        SurveyInfo.ViewOrder =
Convert.ToInt32(ConvertNullInteger(dr("ViewOrder")))
        SurveyInfo.CreatedByUser = Convert.ToInt32(dr("CreatedByUser"))
        SurveyInfo.CreatedDate = Convert.ToDateTime(dr("CreatedDate"))
        SurveyInfolist.Add(SurveyInfo)
    End While
    End Using
    Return SurveyInfolist
End Function

Public Shared Function GetSurvey(ByVal SurveyID As Integer, ByVal ModuleId As
Integer) As SurveyInfo
    Dim SurveyInfo As SurveyInfo = New SurveyInfo
    Using dr As IDataReader =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurvey", SurveyID,
ModuleId)
    While dr.Read
```

```
        SurveyInfo.SurveyId = Convert.ToInt32(dr("SurveyId"))
        SurveyInfo.ModuleId = Convert.ToInt32(dr("ModuleID"))
        SurveyInfo.Question = Convert.ToString(dr("Question"))
        SurveyInfo.OptionType = Convert.ToString(dr("OptionType"))
        SurveyInfo.ViewOrder =
Convert.ToInt32(ConvertNullInteger(dr("ViewOrder")))
        SurveyInfo.Votes = Convert.ToInt32(ConvertNullInteger(dr("Votes")))
        SurveyInfo.CreatedByUser = Convert.ToInt32(dr("CreatedByUser"))
        SurveyInfo.CreatedDate = Convert.ToDateTime(dr("CreatedDate"))
    End While
    End Using
    Return SurveyInfo
End Function

Public Shared Sub DeleteSurvey(ByVal objSurvey As SurveyInfo)
    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("DeleteSurvey",
objSurvey.SurveyId, objSurvey.ModuleId)
End Sub

Public Shared Function AddSurvey(ByVal objSurvey As SurveyInfo) As Integer
    Return
CType(DotNetNuke.Data.DataProvider.Instance().ExecuteScalar("AddSurvey",
objSurvey.ModuleId, objSurvey.Question, GetNull(objSurvey.ViewOrder),
objSurvey.OptionType, objSurvey.CreatedByUser), Integer)
End Function

Public Shared Sub UpdateSurvey(ByVal objSurvey As SurveyInfo)
    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("UpdateSurvey",
objSurvey.SurveyId, objSurvey.Question, GetNull(objSurvey.ViewOrder),
objSurvey.OptionType, objSurvey.CreatedByUser, objSurvey.ModuleId)
End Sub

Public Shared Function GetNull(ByVal Field As Object) As Object
    Return Null.GetNull(Field, DBNull.Value)
End Function
```

SurveyOptionController.vb:

```
Public Shared Function GetSurveyOptions(ByVal SurveyId As Integer) As List(Of
SurveyOptionInfo)
    Dim SurveyOptionInfolist As List(Of SurveyOptionInfo) = New List(Of
SurveyOptionInfo)
    Using dr As IDataReader =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurveyOptions",
SurveyId)
    While dr.Read
```

```
        Dim SurveyOptionInfo As SurveyOptionInfo = New SurveyOptionInfo
        SurveyOptionInfo.SurveyOptionId = Convert.ToInt32(dr("SurveyOptionID"))
        SurveyOptionInfo.OptionName = Convert.ToString(dr("OptionName"))
        SurveyOptionInfo.IsCorrect = Convert.ToString(dr("IsCorrect"))
        SurveyOptionInfo.Votes =
Convert.ToInt32(SurveyController.ConvertNullInteger(dr("Votes")))
        SurveyOptionInfo.ViewOrder =
Convert.ToInt32(SurveyController.ConvertNullInteger(dr("ViewOrder")))
        SurveyOptionInfolist.Add(SurveyOptionInfo)
    End While
    End Using
    Return SurveyOptionInfolist
End Function

Public Shared Sub DeleteSurveyOption(ByVal objSurveyOption As SurveyOptionInfo)

DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("DeleteSurveyOption",
objSurveyOption.SurveyOptionId)
End Sub

Public Shared Function AddSurveyOption(ByVal objSurveyOption As
SurveyOptionInfo) As Integer
    Return
CType(DotNetNuke.Data.DataProvider.Instance().ExecuteScalar("AddSurveyOption",
objSurveyOption.SurveyId, objSurveyOption.OptionName,
SurveyController.GetNull(objSurveyOption.ViewOrder), objSurveyOption.IsCorrect),
Integer)
End Function

Public Shared Sub UpdateSurveyOption(ByVal objSurveyOption As SurveyOptionInfo)

DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("UpdateSurveyOption",
objSurveyOption.SurveyOptionId, objSurveyOption.OptionName,
SurveyController.GetNull(objSurveyOption.ViewOrder), objSurveyOption.IsCorrect)
End Sub

Public Shared Sub AddSurveyResult(ByVal objSurveyOption As SurveyOptionInfo,
ByVal UserID As Integer)
    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("AddSurveyResult",
objSurveyOption.SurveyOptionId, UserID)
End Sub
```

C#

SurveyController.cs:

```
static public List<SurveyInfo> GetSurveys(int ModuleId)
{
   List<SurveyInfo> SurveyInfolist = new List<SurveyInfo>();
   using (IDataReader dr =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurveys", ModuleId))
   {
   while (dr.Read())
   {
      SurveyInfo colSurveyInfo = new SurveyInfo();
      colSurveyInfo.SurveyId = Convert.ToInt32(dr["SurveyId"]);
      colSurveyInfo.Question = Convert.ToString(dr["Question"]);
      colSurveyInfo.OptionType = Convert.ToString(dr["OptionType"]);
      colSurveyInfo.ViewOrder =
Convert.ToInt32(ConvertNullInteger(dr["ViewOrder"]));
      colSurveyInfo.CreatedByUser = Convert.ToInt32(dr["CreatedByUser"]);
      colSurveyInfo.CreatedDate = Convert.ToDateTime(dr["CreatedDate"]);
      SurveyInfolist.Add(colSurveyInfo);
   }
   }
   return SurveyInfolist;
}

static public SurveyInfo GetSurvey(int SurveyID, int ModuleId)
{
   SurveyInfo colSurveyInfo = new SurveyInfo();
   using (IDataReader dr =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurvey",SurveyID,
ModuleId))
   {
   while (dr.Read())
   {
      colSurveyInfo.SurveyId = Convert.ToInt32(dr["SurveyId"]);
      colSurveyInfo.ModuleId = Convert.ToInt32(dr["ModuleID"]);
      colSurveyInfo.Question = Convert.ToString(dr["Question"]);
      colSurveyInfo.OptionType = Convert.ToString(dr["OptionType"]);
      colSurveyInfo.ViewOrder =
Convert.ToInt32(ConvertNullInteger(dr["ViewOrder"]));
      colSurveyInfo.Votes = Convert.ToInt32(ConvertNullInteger(dr["Votes"]));
      colSurveyInfo.CreatedByUser = Convert.ToInt32(dr["CreatedByUser"]);
      colSurveyInfo.CreatedDate = Convert.ToDateTime(dr["CreatedDate"]);
   }
   }
   return colSurveyInfo;
}

public static void DeleteSurvey(SurveyInfo objSurvey)
{
```

```
    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("DeleteSurvey",
objSurvey.SurveyId, objSurvey.ModuleId);
}

public static int AddSurvey(SurveyInfo objSurvey)
{
    return
(Convert.ToInt32(DotNetNuke.Data.DataProvider.Instance().ExecuteScalar("AddSurv
ey", objSurvey.ModuleId, objSurvey.Question, GetNull(objSurvey.ViewOrder),
objSurvey.OptionType, objSurvey.CreatedByUser)));
}

public static void UpdateSurvey(SurveyInfo objSurvey)
{
    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("UpdateSurvey",
objSurvey.SurveyId, objSurvey.Question, GetNull(objSurvey.ViewOrder),
objSurvey.OptionType, objSurvey.CreatedByUser, objSurvey.ModuleId);
}

public static object GetNull(object Field)
{
    return DotNetNuke.Common.Utilities.Null.GetNull(Field, DBNull.Value);
}
```

SurveyOptionController.cs:

```
public class SurveyOptionController
{
    static public List<SurveyOptionInfo> GetSurveyOptions(int SurveyId)
    {
    List<SurveyOptionInfo> SurveyOptionInfolist = new List<SurveyOptionInfo>();
    using (IDataReader dr =
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurveyOptions",Surve
yId))
    {
    while (dr.Read())
    {
        SurveyOptionInfo colSurveyOptionInfo = new SurveyOptionInfo();
        colSurveyOptionInfo.SurveyOptionId =
Convert.ToInt32(dr["SurveyOptionID"]);
        colSurveyOptionInfo.OptionName = Convert.ToString(dr["OptionName"]);
        colSurveyOptionInfo.IsCorrect = Convert.ToBoolean(dr["IsCorrect"]);
        colSurveyOptionInfo.Votes =
Convert.ToInt32(SurveyController.ConvertNullInteger(dr["Votes"]));
        colSurveyOptionInfo.ViewOrder =
Convert.ToInt32(SurveyController.ConvertNullInteger(dr["ViewOrder"]));
```

```
            SurveyOptionInfolist.Add(colSurveyOptionInfo);
        }
        }
        return SurveyOptionInfolist;
    }

    public static void DeleteSurveyOption(SurveyOptionInfo objSurveyOption)
    {

    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("DeleteSurveyOption",o
    bjSurveyOption.SurveyOptionId);
    }

    public static int AddSurveyOption(SurveyOptionInfo objSurveyOption)
    {
        return
    (Convert.ToInt32(DotNetNuke.Data.DataProvider.Instance().ExecuteScalar("AddSurv
    eyOption", objSurveyOption.SurveyId, objSurveyOption.OptionName,
    SurveyController.GetNull(objSurveyOption.ViewOrder),
    objSurveyOption.IsCorrect)));
    }

    public static void UpdateSurveyOption(SurveyOptionInfo objSurveyOption)
    {

    DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("UpdateSurveyOption",
    objSurveyOption.SurveyOptionId, objSurveyOption.OptionName,
    SurveyController.GetNull(objSurveyOption.ViewOrder), objSurveyOption.IsCorrect);
    }

    public static void AddSurveyResult(SurveyOptionInfo objSurveyOption, int UserID)
    {
        DotNetNuke.Data.DataProvider.Instance().ExecuteNonQuery("AddSurveyResult",
    objSurveyOption.SurveyOptionId, UserID);
    }
```

What Did We Just Do?

Essentially we are replacing code such as this:

```
DataProvider.Instance().GetSurveys(ModuleId)
```

with this:

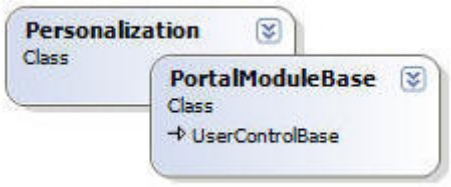
```
DotNetNuke.Data.DataProvider.Instance().ExecuteReader("GetSurveys", ModuleId)
```

In doing so we are able to eliminate the need to create an abstract provider and a concrete provider. This saves hundreds of lines of code and potentially hours of development.

In addition, the DAL+ allows you to connect to the database with a single line of code. You do not have to set the database connection parameters yourself.

Using the DotNetNuke API (Application Programming Interface)

Module Settings and Personalization: Easily Store Data for Your Module

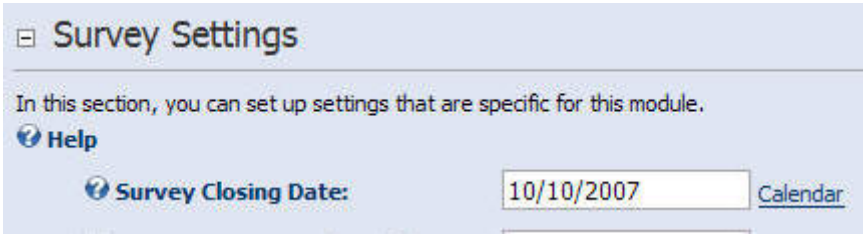


Nearly half of your module development can be spent creating database objects (tables and stored procedures) and data access methods. You can leverage the DotNetNuke framework and reduce or eliminate this task.

What Module Settings Will Do For You

Module Settings will allow you to easily store data that is specific to an instance of a module. A DotNetNuke module can be placed on multiple pages in a portal (even on the same page). Each time is a separate instance.

Module Settings is usually used to store the data for the **Settings** user control of a module, but it can be used on any of the module's user controls.



What Personalization Will Do For You

Personalization will store data that is specific to a single user. With the DotNetNuke Survey module, personalization is used to determine if an authenticated user has voted before. If they have they are only showed the results and not allowed to vote again.

Implementing Module Settings

We used this code to save data to the Module Settings:

VB:

```
Dim objModules As New DotNetNuke.Entities.Modules.ModuleController
objModules.UpdateModuleSetting(ModuleId, "surveyclosingdate",
DotNetNuke.Common.Globals.DateToString(datClosingDate))
```

C#:

```
DotNetNuke.Entities.Modules.ModuleController objModules = new
DotNetNuke.Entities.Modules.ModuleController();
objModules.UpdateModuleSetting(ModuleId, "surveyclosingdate",
DotNetNuke.Common.Globals.DateToString(datClosingDate));
```

We retrieve the data using this code:

VB:

```
txtClosingDate.Text = CType(ModuleSettings("surveyclosingdate"),
Date).ToShortDateString
```

C#:

```
txtClosingDate.Text =
Convert.ToDateTime(ModuleSettings["surveyclosingdate"]).ToShortDateString()
;
```

Implementing Personalization

To implement Personalization this code is used:

VB:

```
DotNetNuke.Services.Personalization.Personalization.SetProfile(ModuleId.ToStr
ing, "Voted", True)
```

C#:

```
DotNetNuke.Services.Personalization.Personalization.SetProfile(ModuleId.ToStr
ing(), "Voted", true);
```

We retrieve the Personalization setting using this code:

VB:

```
' Check if the user has voted before
If Not
DotNetNuke.Services.Personalization.Personalization.GetProfile(ModuleId.ToStr
ing, "Voted") Is Nothing Then
   Return
CType(DotNetNuke.Services.Personalization.Personalization.GetProfile(ModuleI
d.ToString, "Voted"), Boolean)
Else
   Return False
End If
```

C#:

```
// Check if the user has voted before
if
(!(DotNetNuke.Services.Personalization.Personalization.GetProfile(ModuleId.To
String(), "Voted") == null))
{
 return
((bool)(DotNetNuke.Services.Personalization.Personalization.GetProfile(Module
Id.ToString(), "Voted")));
}
```

```
  else
{
  return false;
}
```

Leverage the DotNetNuke framework

Generally you will want to create tables and stored procedures and data access methods so that you can optimize performance. However, many situations may exist where minimal amounts of data are needed and development time can be reduced by leveraging the DotNetNuke framework.

Localization: Easily Translate Your Module

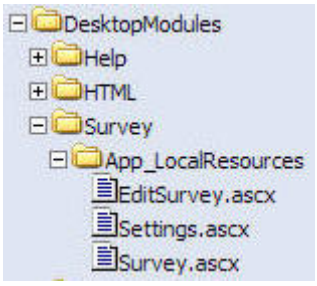
An advantage to using the DotNetNuke framework is it's extensive support for **Localization**. This provides the ability to translate the content of a portal. The key to making this work is that the module programmer must allow the module to be **localized**.

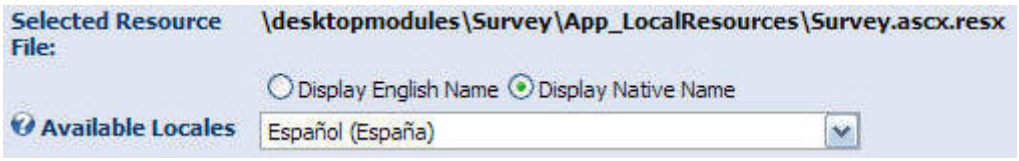
What Localization Will Do For You

A typical scenario shows the steps a portal administrator would make to provide a Spanish translation for users of their portal.

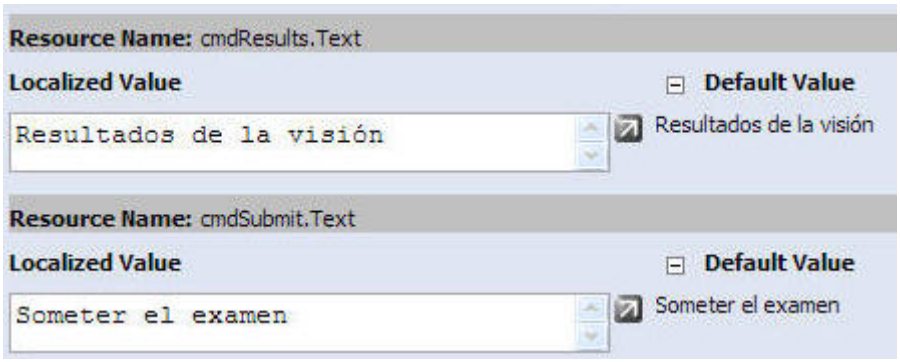
1. The Portal administrator adds a ***Locale*** from the *Admin -> Languages* menu

2. The Portal administrator uses the Language Editor and selects the User Control to be localized and chooses the Locale.



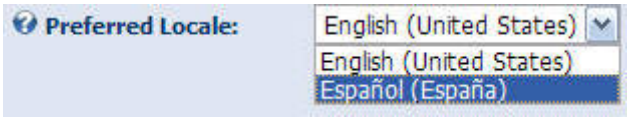


3. The administrator then enters translations in the file:

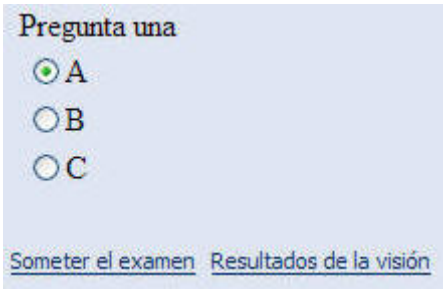


4. The Visitors to the Portal can select **Manage Profile**

5. And change their **Preferred Locale**



6. And the module will display in the chosen language.



Implementing Localization for the Survey Module

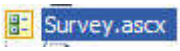
Localizing Static Content

In the Survey module, we ensured that various text elements were Localized by using labels rather than text. All other controls such as link buttons are easily localized. Link buttons such as this:

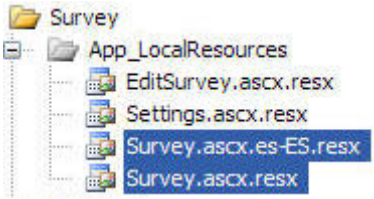
Are coded with a unique resourcekey.

```
<asp:linkbutton id="cmdSubmit" runat="server" resourcekey="cmdSubmit"
cssclass="CommandButton">Submit Survey</asp:linkbutton>
<asp:linkbutton id="cmdResults" runat="server" resourcekey="cmdResults"
cssclass="CommandButton">View Results</asp:linkbutton>
```

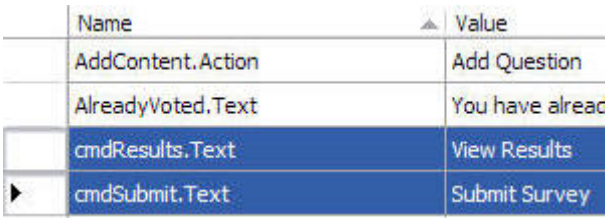
The User Control...



Is accompanied by resource files that begin with the same name and reside in the **App_LocalResouces** directory (that resides in the module folder).



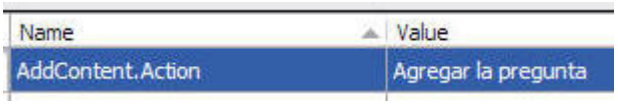
The resource file contains the resource key and the value that will be displayed.



Localizing Dynamic Content

For the text that is generated dynamically such as the Administrator menu...

The same resource file is used.



This code is used to return the localized value:

```
Localization.GetString(Entities.Modules.Actions.ModuleActionType.AddContent,
LocalResourceFile)
```

NavigateURL: How to make a link

To navigate from the **View control** to the **Edit control** you can use code such as this:

```
Response.Redirect(EditUrl())
```

To navigate from the **Edit control** to the **View control** you can use code such as this:

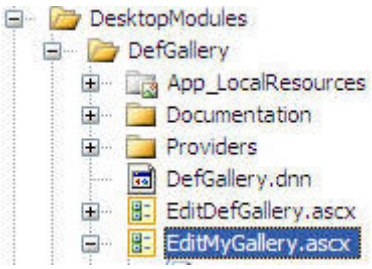
```
Response.Redirect(Globals.NavigateURL(), true)
```

You usually don't need a link for the **Settings** control. You simply configure the User Control in in the module definition as a *Settings control* and a link is automatically created in the module's menu to navigate to it.

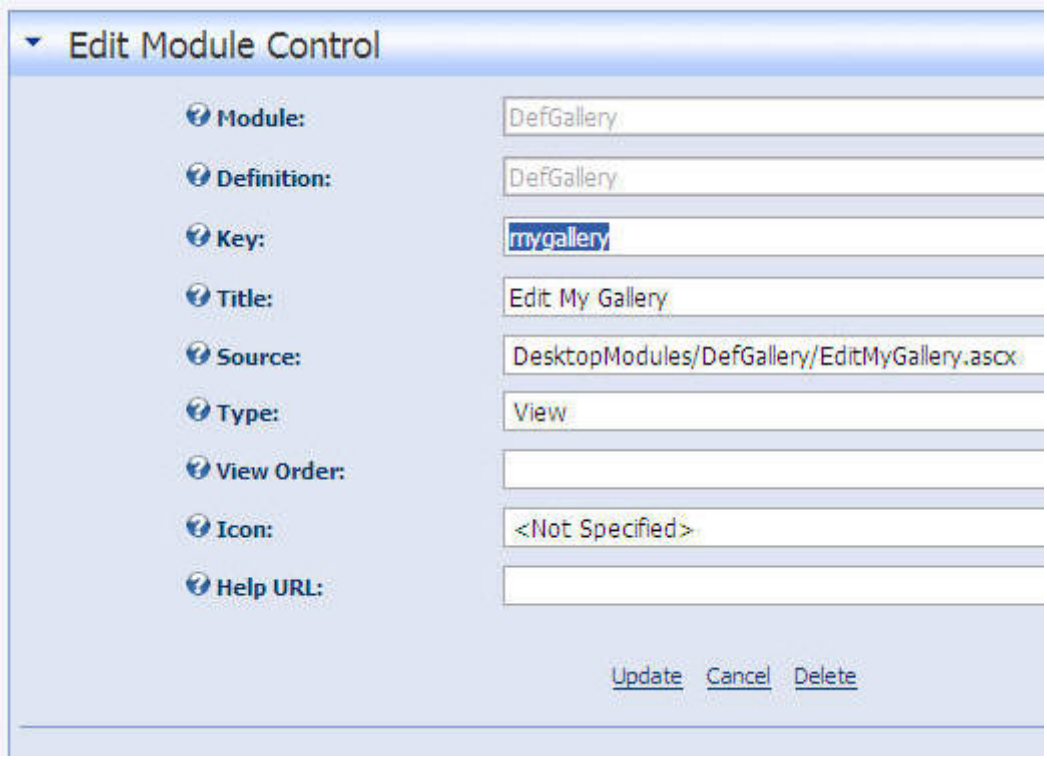
Navigating to a User Control in Your Module

If you want to navigate to User Control other than the default controls covered above, you have to perform a few steps.

1) Create the User Control. In this example we have created a User Control called *EditMyGallery.ascx*.



2) Add the User Control to the Module definition and give it a unique key.

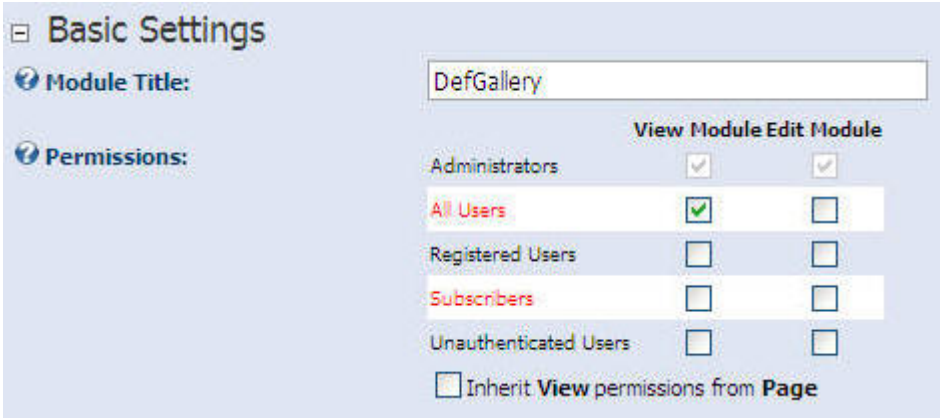


The unique key we have given this control is ***mygallery*.**

Notice the Module name is **DefGallery**. This is important and will be used in the next step.

The Type has been set to **View.** In the example module we intend for the *EditMyGallery.ascx* control to be accessible by all users who have access to the View controls.

However, realize that the Portal administrator is able to configure the security access for the different types of controls. The Portal administrator usually configures the module to allow all users to access the View controls (even users who have not logged in).



However, this is not always the case. The Portal may be a private portal that requires a person to be logged in to use the module. In that case the Portal administrator will configure the View controls for the module to be accessible only by *Registered Users*.

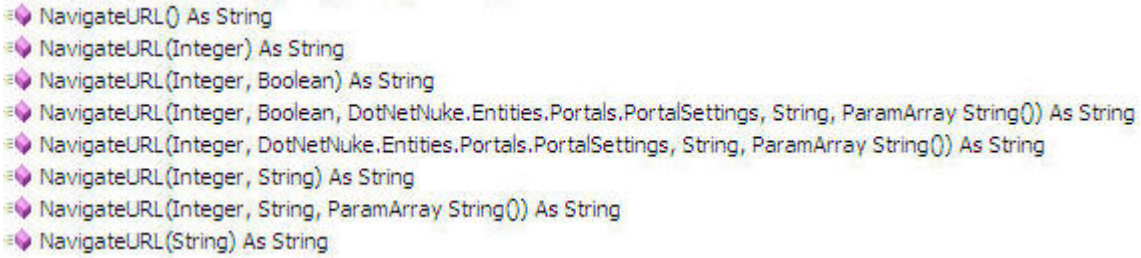
You simply indicate a *security group* for the control. The Portal administrator will ultimately decide which users actually have access to the control.

VERY IMPORTANT

In the Module definition, you must set the **Default Cache Time** to ***0*** (and click the **Update Cache Properties** button to set it). Otherwise your links may not work properly because of module caching.

Using NavigateURL

Now that your module configuration is set you can use the NavigateURL method. When you look at the NavigateURL method in the object browser you can see that it has a number of overloads



The overload that will be used in this example is:

```
Public Function NavigateURL
(
ByVal TabID As Integer,
ByVal ControlKey As String,
ByVal ParamArray AdditionalParameters() As String
)
As String
```

(***note:*** *you will want to add* ***DotNetNuke.Common*** *to the* ***Imports*** statement *(for VB.NET) or* ***using*** *statement (for C#) to use the code below*)

You can use the following code to navigate to the User Control:

C#:

```
protected void cmdEdit_Click(object sender, EventArgs e)
{
ModuleController objModules = new ModuleController();
Response.Redirect(Globals.NavigateURL(PortalSettings.ActiveTab.TabID,
"mygallery", "mid=" + ModuleId.ToString()));
}
```

VB:

```
Protected Sub cmdEdit_Click(ByVal sender As Object, ByVal e As EventArgs)
Dim objModules As ModuleController = New ModuleController
Response.Redirect(Globals.NavigateURL(PortalSettings.ActiveTab.TabID,
"mygallery", "mid=" & Cstr(ModuleID)))
End Sub
```

This code creates a URL such as:

```
http://localhost/MyDNNWebsite/Home/tabid/36/ctl/mygallery/mid/362/Defau
lt.aspx
```

The code:

```
PortalSettings.ActiveTab.TabID
```

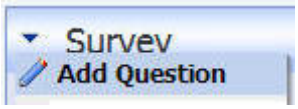
Inserts *tabid/36* into the URL so that the DotNetNuke framework knows which page to go to (in this example we want to go to the current page).

The "mygallery" parameter inserts *mygallery* after *ctl*. This instructs the DotNetNuke framework to load the User Control that is configured with the *mygallery* key. The code:

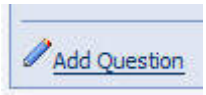
```
"mid=" + ModuleId.ToString()
```

inserts *mid/362* into the URL so that the DotNetNuke framework knows which module to load.

IActionable: Add Items to Your Module Menu



The DotNetNuke Survey module displays an *Add Question* option on the module's menu (and in the footer of the module) for any user that has *Edit* access to the module (usually the Administrator).



What IActionable Will Do For You

If you are familiar with the **NavigateURL** function you know how to provide navigation from one user control to another in your modules. To add items to the modules menu, you use the IActionable interface.

Implementing IActionable

To implement IActionable, this code is used:

VB:

```
Partial Class Survey
Implements Entities.Modules.IActionable

Public ReadOnly Property ModuleActions() As
DotNetNuke.Entities.Modules.Actions.ModuleActionCollection Implements
DotNetNuke.Entities.Modules.IActionable.ModuleActions
Get

Dim Actions As New Entities.Modules.Actions.ModuleActionCollection
    Actions.Add(
    GetNextActionID,
    Localization.GetString(Entities.Modules.Actions.ModuleActionType.AddContent,
LocalResourceFile),
    Entities.Modules.Actions.ModuleActionType.AddContent,
```

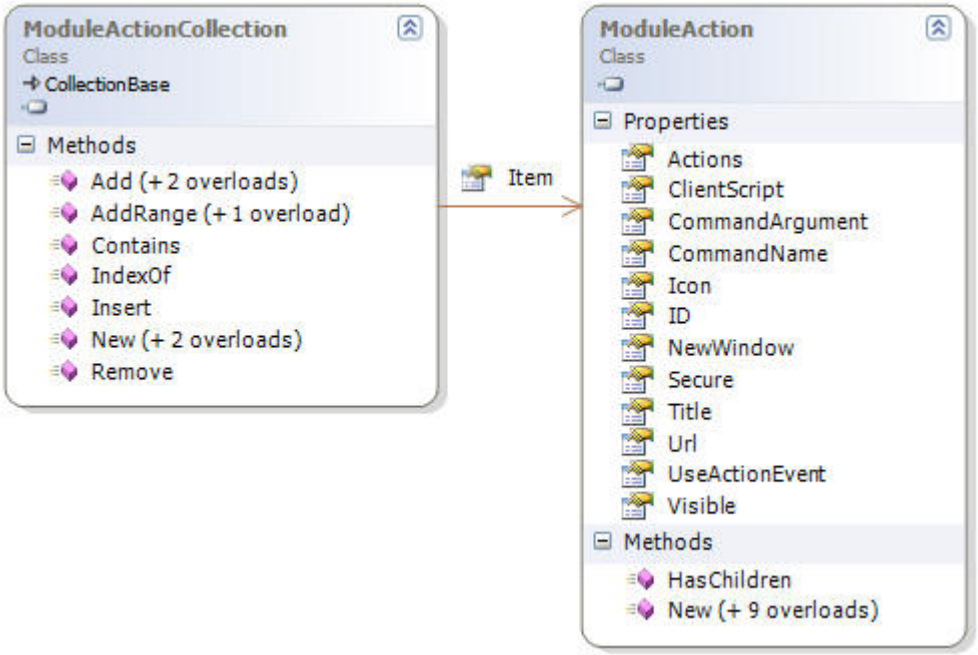
```
    "",
    "",
    EditUrl(),
    False,
    SecurityAccessLevel.Edit,
    True,
    False
    )

Return Actions
End Get
End Property
```

C#:

```
public partial class Survey : Entities.Modules.IActionable

public DotNetNuke.Entities.Modules.Actions.ModuleActionCollection ModuleActions
{
get
{

Entities.Modules.Actions.ModuleActionCollection Actions = new
Entities.Modules.Actions.ModuleActionCollection();
  Actions.Add(GetNextActionID(),
  Localization.GetString(Entities.Modules.Actions.ModuleActionType.AddContent,
  LocalResourceFile),
  Entities.Modules.Actions.ModuleActionType.AddContent,
  "",
  "",
  EditUrl(),
  false,
  SecurityAccessLevel.Edit,
  true, false);
  return Actions;

}
}
```

What Did We Just Do?

When we implement the IActionable interface we communicate to the DotNetNuke framework that we will create a property that will return a ModuleActionCollection object. We can see in the class diagram below that ModuleActionCollection is a collection of ModuleAction objects.



To create a ModuleActionCollection collection you add a ModuleAction object to it using the Add method of ModuleActionCollection. The Add method of ModuleActionCollection contains a number of overloads. The overload we used is:

Add(
ByVal *ID* As **Integer**,
ByVal *Title* As **String**,
ByVal *CmdName* As **String**,
Optional ByVal *CmdArg* As **String** = "",
Optional ByVal *Icon* As **String** = "",
Optional ByVal *Url* As **String** = "",
Optional ByVal *UseActionEvent* As **Boolean** = False,
Optional ByVal *Secure* As **DotNetNuke**.**Security**.**SecurityAccessLevel** = Anonymous,

Optional ByVal *Visible* As **Boolean** = True,
Optional ByVal *NewWindow* As **Boolean** = False
) As **DotNetNuke**.**Entities**.**Modules**.**Actions**.**ModuleAction**

This overload creates a ModuleAction object and adds it to the ModuleActionCollection collection.

The following table explains the meaning of each field and shows sample values and data:

Property	Value	Sample Data
ID As **Integer** **[Each module action must have a unique ID. Use the** GetNextActionID **method to generate a unique id]**	GetNextActionID	3
Title As **String** **[Sets the text displayed on the menu]**	Localization.GetString(Entities.Modules.Actions .ModuleActionType.AddContent, LocalResourceFile)	"Add Question"
CmdName As **String** **[The command name passed to the client when this action is clicked. Used for JavaScript]**	Entities.Modules.Actions.ModuleActionType.AddContent	"AddContent.Action"
CmdArg As **String** **[The command argument passed to the client when this action is clicked. Used for additional parameters passed to JavaScript]**	""	""
Icon As **String** **[The URL of the Icon to place next to this action]**	""	""
Url As **String** **[The destination URL to redirect the client browser when this action is clicked.]**	EditUrl()	"http://localhost/DotNetNuke/Home/tabid/36/ctl/Edit/mid/423/Default.aspx"
UseActionEvent As **Boolean** **[Determines whether client will receive an event notification]**	False	

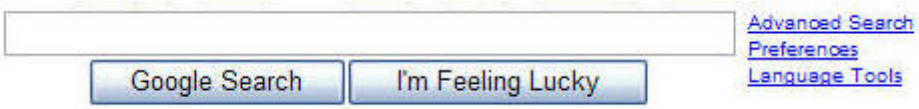
Secure As **DotNetNuke.Security.SecurityAccessLevel** **[The security access level required for access to this action]**	SecurityAccessLevel.Edit	1
Visible As **Boolean** **[Whether this action will be displayed]**	True	True
NewWindow As **Boolean** **[Whether this action will be displayed in a new window]**	False	False

ISearchable: Easily Make Your Module Searchable

The ISearchable interface is used to allow the users of your module to search for content using the search mechanism provided by the DotNetNuke framework.

What ISearchable Will Do For You

Most uses are familiar with the search provided by websites such as Google.com



If you type in "*create a DotNetNuke module*" in Google you will see a list of websites that describe how to make DotNetNuke modules. If you type in "*buy a DotNetNuke module*" you will see sites that sell DotNetNuke modules. Google is able to understand (to the best of its ability) what you are ultimately trying to find by searching through the entire contents of all the pages it has stored in its database.

The search provided by the DotNetNuke framework doesn't work like that. It works more like an index in the back of a book. The only items in the index are items that the author (in this case the module developer) has decided to put there. The search merely allows DotNetNuke portal users to quickly find items placed in this index.

The DotNetNuke Search is Very Useful

At first glance you might prefer the Google approach and not see the usefulness of the search provided by the DotNetNuke framework. However, like a carefully constructed index in a book, the DotNetNuke search allows the module developer to deliver helpful relevant results to search queries by only placing items in that index that a user would want to search on. In this case what you leave out is as important as what you put in.

For the Survey module it was decided that only the questions not the answers would be placed in the search index. This omitted such potentially repetitive and unhelpful words such as *yes* and *no* from cluttering up the search results.

Implementing Search for the Survey Module

To implement search for the Survey module we performed three steps:

- Indicate that the *controller* class will implement the ISearchable interface
- Insert the code for the ISearchable interface
- Update the module configuration

Implement ISearchable in the Controller Class

When you look at the module definition for the Survey module, you can see that the controller class defined is *DotNetNuke.Modules.Survey.SurveyController*.

We opened up this class in the Visual Studio code editor and added this line at the top of the class and hit the **Enter** key.

VB:

```
Implements Entities.Modules.ISearchable
```

C#:

```
: Entities.Modules.ISearchable
```

Hitting the **Enter** key after we insert the line causes Visual Studio to insert a *stub* for the method.

Next, we add the following code to the method and save the page.

VB:

```
Public Function GetSearchItems(ByVal ModInfo As Entities.Modules.ModuleInfo) _
As Services.Search.SearchItemInfoCollection Implements
Entities.Modules.ISearchable.GetSearchItems
' Get the Surveys for this Module instance
Dim colSurveys As List(Of SurveyInfo) = GetSurveys(ModInfo.ModuleID)
Dim SearchItemCollection As New SearchItemInfoCollection
Dim SurveyInfo As SurveyInfo
For Each SurveyInfo In colSurveys
Dim SearchItem As SearchItemInfo
SearchItem = New SearchItemInfo _
```

```
(ModInfo.ModuleTitle & " - " & SurveyInfo.Question, _
SurveyInfo.Question, _
SurveyInfo.CreatedByUser, _
SurveyInfo.CreatedDate, ModInfo.ModuleID, _
SurveyInfo.SurveyId, _
SurveyInfo.Question)
SearchItemCollection.Add(SearchItem)
Next
Return SearchItemCollection
End Function
```

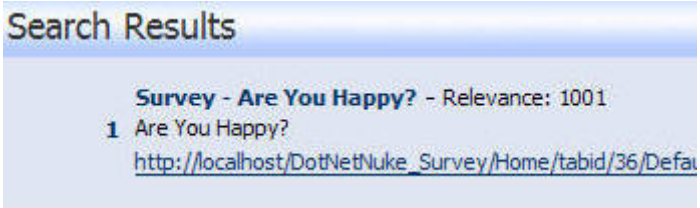
C#:

```
public SearchItemInfoCollection GetSearchItems(DotNetNuke.Entities.Modules.ModuleInfo
ModInfo)
{
// Get the Surveys for this Module instance
List<SurveyInfo> colSurveys = GetSurveys(ModInfo.ModuleID);
SearchItemInfoCollection SearchItemCollection = new SearchItemInfoCollection();
foreach (SurveyInfo SurveyInfo in colSurveys)
{
SearchItemInfo SearchItem;
SearchItem = new SearchItemInfo
(ModInfo.ModuleTitle + " - " + SurveyInfo.Question,
SurveyInfo.Question,
SurveyInfo.CreatedByUser,
SurveyInfo.CreatedDate, ModInfo.ModuleID,
Convert.ToString(SurveyInfo.SurveyId),
SurveyInfo.Question);
SearchItemCollection.Add(SearchItem);
}
return SearchItemCollection;
}
```

We return to the module definition (for the Survey module) and click the *Update* link.

Searchable will now be checked under **Supported Features**.

The content will now be searchable.



To implement the search we performed three steps:

- Created and filled a **SearchItemInfo** object
- Added this object to the **SearchItemInfoCollection** collection
- Returned the **SearchItemInfoCollection** as the output for the method

If you look in the object browser at the definition for the **SearchItemInfo** object you can see that it has a number of overloads.

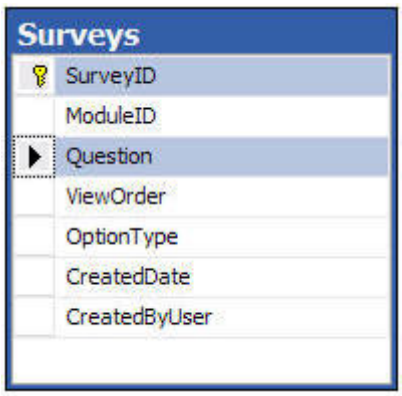
The constructor we used has this signature:

```
Public Sub New(
ByVal Title As String,
ByVal Description As String,
ByVal Author As Integer,
```

```
ByVal PubDate As Date,
ByVal ModuleID As Integer,
ByVal SearchKey As String,
ByVal Content As String
)
Member of: DotNetNuke.Services.Search.SearchItemInfo
```

The important thing to remember is that the *SearchKey* parameter must be a unique value. In this case we passed the contents of the **SurveyId** field (from the Surveys table) to the *SearchKey* parameter.

The *Content* is the content that the portal users will be searching on. We passed the contents of the **Question** field to the *Content* parameter. As you can see in the table schema below, the **Question** field has a direct one-to-one relationship with the **SurveyID** field.



This line adds the **SearchItemInfo** object to the **SearchItemInfoCollection** collection:

```
SearchItemCollection.Add(SearchItem)
```

This line returns the **SearchItemInfoCollection** collection as the output of the method:

```
Return SearchItemCollection
```

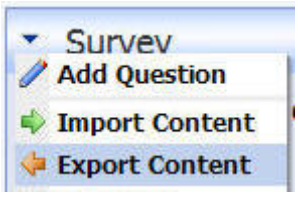
IPortable: Easily export content from your module to deploy on your production server

The IPortable interface will allow the users of your module to develop content on a development server and easily deploy this content to a production server. You will want to implement this interface in practically every module you create no matter how simple it is. The good news is that this task is very easy to do and does not require advanced programming or a significant amount of time.

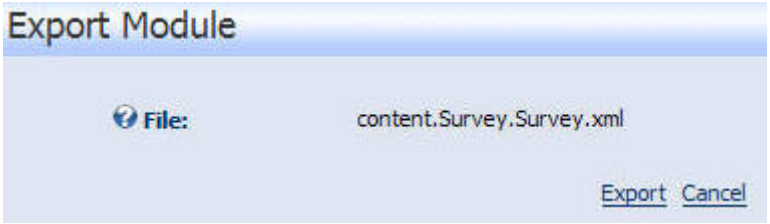
What IPortable Will Do For You

For example if you have a development server at your company and you created a survey using the Survey module and you now want to put the Survey on your production server. To do so you would simply follow these steps:

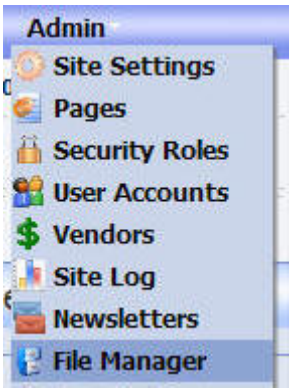
1) From the menu of the Survey module select **Export Content**



2) Next, from the **Export Module** page click the **Export** button

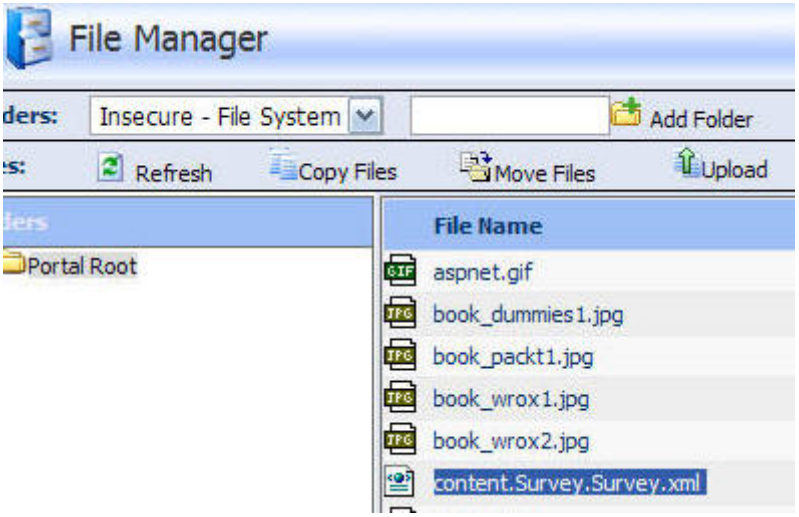


3) While logged in as the portal Administrator select **File Manger** from the **Admin** menu



*(do this while logged in as portal Administrator not as the portal Host using the Host account because you will not see the correct directory in the **File Manager**)*

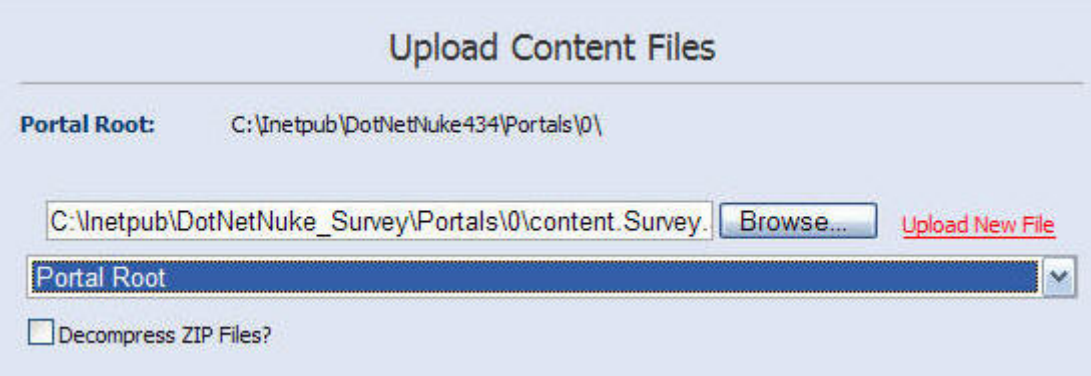
We see the .xml file that has been created (in this example *content.Survey.Survey.xml*



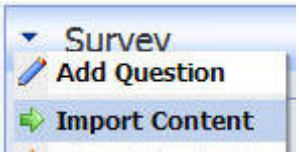
4) Import the .xml file to your production server using these steps:

- Log in as the Administrator of the portal
- From the **Admin** menu select **File Manger**
- In the **File Manager** click **Upload**

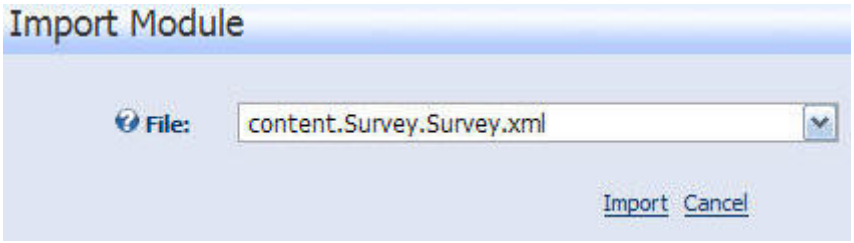
Browse to the location of the .xml file and after ensuring that **Portal Root** is selected in the location drop-down, click **Upload New File**



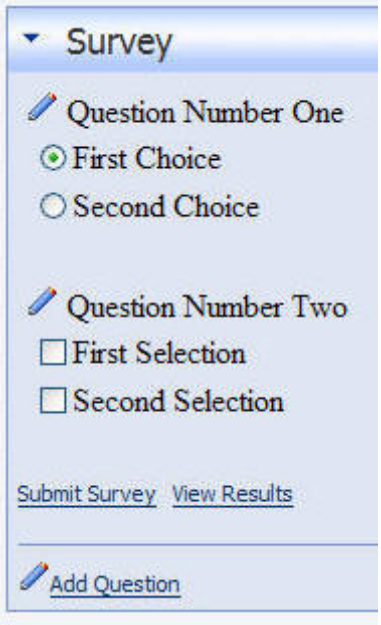
5) Next place an instance of the Survey module on a page and from the Survey module's menu select **Import Content**



6) The .xml file is show in the drop-down. Click the **Import** button



And you're done.



Implementing IPortable

To implement this functionality in your own custom DotNetNuke module you simply:

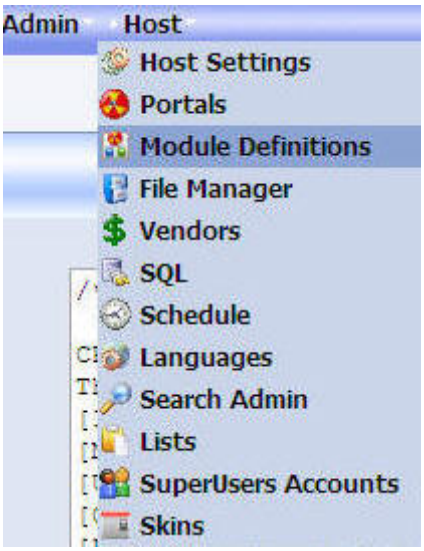
- Specify a **business controller class** in the module definition for the module
- Add the line ***Implements Entities.Modules.IPortable*** in the **business controller class**
- Create a **ExportModule** method and an **ImportModule** method

Specify a business controller class

The first thing you need to do is indicate a business controller class in the module definition for your module

Here are the steps we used to create the module definition for the Survey module:

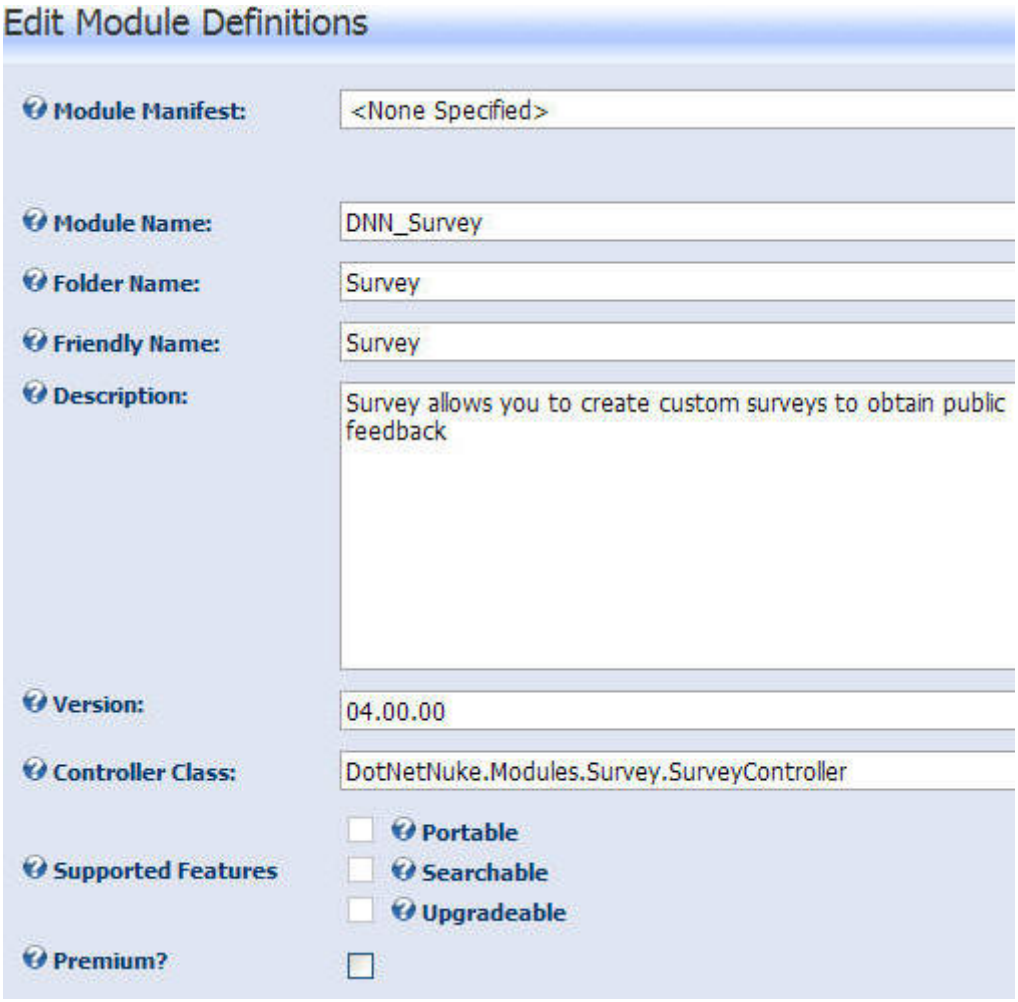
While logged into the DotNetNuke site as the "**host**" account, from the menu bar we selected **Host** and then selected **Module Definitions**



We click on the black arrow that is pointing down to make the fly-out menu to appear. On that menu we select **Create New Module**

In the **Edit Module Definitions** page we enter:

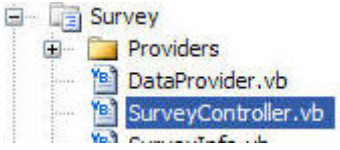
- *DNN_Survey* for **MODULE NAME**
- *Survey* for **FOLDER TITLE**
- Survey for **FRIENDLY TITLE**
- Survey allows you to create custom surveys to obtain public feedback for **DESCRIPTION**
- 04.00.00 for **VERSION**
- DotNetNuke.Modules.Survey.SurveyController for **Controller Class**



We then click **UPDATE** and completed the module definition

Add the Line *Implements Entities.Modules.IPortable* to the Business Controller Class

We created a **SurveyController.vb** class file



In that class we added the line *Implements Entities.Modules.IPortable*

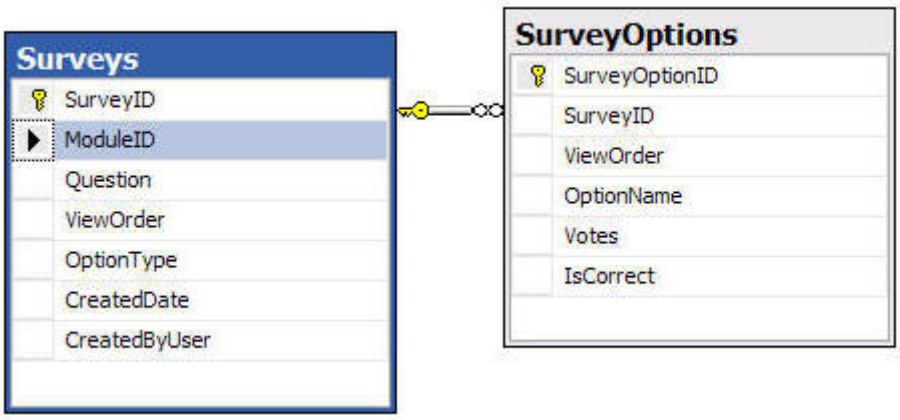
```
29
30  Namespace DotNetNuke.Modules.Survey
31
32      Public Class SurveyController
33          Implements DotNetNuke.Entities.Modules.I
34          Implements Entities.Modules.ISearchable
35          Implements Entities.Modules.IPortable
36
```

Visual Studio then automatically adds the two *stub* methods.

Create a *ExportModule* method

When the users of your module selects **Export Content**, the DotNetNuke framework will call the **ExportModule** method that you create and pass it a **ModuleID** parameter. It is expecting your method to return an .xml formatted response as a String data type.

When implementing this for the Survey module we can see that the Surveys are contained in the **Surveys** table, but the options (for example "*First Choice*") are contained in the **SurveyOptions** table.



For each Survey in the **Surveys** table, we needed to export the matching options for that survey in the **SurveyOptions** table. Another thing to notice is that while the primary key for the **Surveys** table is **SurveyID**, the field that is important right now is the **ModuleID** field. In the DotNetNuke architecture the **ModuleID** is an important

parameter. A **ModuleID** represents a single instance of a module and is used to segment the data for each instance of that module in the database. A **ModuleID** is unique across all portals in a DotNetNuke installation.

Here is our implementation of the **ExportModule** method:

VB:

```
Public Function ExportModule(ByVal ModuleID As Integer) As String Implements
DotNetNuke.Entities.Modules.IPortable.ExportModule

Dim strXML As New StringBuilder()
Dim settings As New XmlWriterSettings()
settings.Indent = True
settings.OmitXmlDeclaration = True
Dim Writer As XmlWriter = XmlWriter.Create(strXML, settings)

'Outer Loop - To build the Surveys
Dim colSurveys As List(Of SurveyInfo) = GetSurveys(ModuleID)
If colSurveys.Count > 0 Then
Writer.WriteStartElement("surveys")
Dim SurveyInfo As SurveyInfo
For Each SurveyInfo In colSurveys
Writer.WriteStartElement("survey")
Writer.WriteElementString("question", SurveyInfo.Question)
Writer.WriteElementString("vieworder", SurveyInfo.ViewOrder)
Writer.WriteElementString("createdbyuser", SurveyInfo.CreatedByUser)
Writer.WriteElementString("createddate", SurveyInfo.CreatedDate)
Writer.WriteElementString("optiontype", SurveyInfo.OptionType.ToString)

   'Inner Loop - To build the Options for each Survey
   Dim colSurveyOptions As List(Of SurveyOptionInfo) =
SurveyOptionController.GetSurveyOptions(SurveyInfo.SurveyId)
   If colSurveyOptions.Count > 0 Then
   Writer.WriteStartElement("surveyoptions")
   Dim SurveyOptionInfo As SurveyOptionInfo
   For Each SurveyOptionInfo In colSurveyOptions
   Writer.WriteStartElement("surveyoption")
   Writer.WriteElementString("optionname", SurveyOptionInfo.OptionName)
   Writer.WriteElementString("iscorrect", SurveyOptionInfo.IsCorrect)
   Writer.WriteElementString("vieworder", SurveyOptionInfo.ViewOrder)
   Writer.WriteEndElement()
   Next ' Retrieve the next SurveyOption
   Writer.WriteEndElement()
   End If
   Writer.WriteEndElement()
```

```
Next
Writer.WriteEndElement()
Writer.Close()

Else
' There is nothing to export
Return String.Empty
End If
Return strXML.ToString()

End Function
```

C#:

```
string DotNetNuke.Entities.Modules.IPortable.ExportModule(int ModuleID)
{
StringBuilder strXML = new StringBuilder();
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.OmitXmlDeclaration = true;
XmlWriter Writer = XmlWriter.Create(strXML, settings);

// Outer Loop - To build the Surveys
List<SurveyInfo> colSurveys = GetSurveys(ModuleID);
if ((colSurveys.Count > 0))
{
Writer.WriteStartElement("surveys");
foreach (SurveyInfo colSurveyInfo in colSurveys)
{
Writer.WriteStartElement("survey");
Writer.WriteElementString("question", colSurveyInfo.Question);
Writer.WriteElementString("vieworder", colSurveyInfo.ViewOrder.ToString());
Writer.WriteElementString("createdbyuser", colSurveyInfo.CreatedByUser.ToString());
Writer.WriteElementString("createddate", colSurveyInfo.CreatedDate.ToShortDateString());
Writer.WriteElementString("optiontype", colSurveyInfo.OptionType.ToString());

  // Inner Loop - To build the Options for each Survey
  List<SurveyOptionInfo> colSurveyOptions =
SurveyOptionController.GetSurveyOptions(colSurveyInfo.SurveyId);
  if ((colSurveyOptions.Count > 0))
  {
  Writer.WriteStartElement("surveyoptions");
  foreach (SurveyOptionInfo colSurveyOptionInfo in colSurveyOptions)
  {
    Writer.WriteStartElement("surveyoption");
    Writer.WriteElementString("optionname", colSurveyOptionInfo.OptionName);
    Writer.WriteElementString("iscorrect", colSurveyOptionInfo.IsCorrect.ToString());
    Writer.WriteElementString("vieworder", colSurveyOptionInfo.ViewOrder.ToString());
```

```
    Writer.WriteEndElement();
  }

// Retrieve the next SurveyOption
Writer.WriteEndElement();
  }
    Writer.WriteEndElement();
  }
Writer.WriteEndElement();
Writer.Close();
}
else
{

// There is nothing to export
return String.Empty;
}
return strXML.ToString();
}
```

Here is a sample of the output created by the DotNetNuke framework from the output of this method:

```
<?xml version="1.0" encoding="utf-8" ?><content type="Survey"
version="04.00.30"><surveys>
 <survey>
  <question>Test Question</question>
  <vieworder>0</vieworder>
  <createdbyuser>1</createdbyuser>
  <createddate>11/11/2006 6:08:26 AM</createddate>
  <optiontype>R</optiontype>
  <surveyoptions>
   <surveyoption>
    <optionname>A</optionname>
    <iscorrect>False</iscorrect>
    <vieworder>0</vieworder>
   </surveyoption>
   <surveyoption>
    <optionname>B</optionname>
    <iscorrect>False</iscorrect>
    <vieworder>1</vieworder>
   </surveyoption>
   <surveyoption>
```

```
    <optionname>C</optionname>
    <iscorrect>False</iscorrect>
    <vieworder>2</vieworder>
   </surveyoption>
  </surveyoptions>
 </survey></surveys></content>
```

Create a *ImportModule* method

When the users of your module select **Import Content**, the DotNetNuke framework will call the **ImportModule** method that you create and pass it a **ModuleID** parameter and a **Content** parameter as well as **Version** and **UserID**. It expects your controller class to process the contents of the **Content** parameter which has a string data type. The **Content** parameter will contain the data in the form of the sample above. Your **ImportModule** method will not return anything. Your method will usually just insert the data passed to in in the **Content** parameter into the database.

Here is our implementation of the **ImportModule** method:

VB:

```
Public Sub ImportModule(ByVal ModuleID As Integer, ByVal Content As String, ByVal Version As
String, ByVal UserId As Integer) Implements
DotNetNuke.Entities.Modules.IPortable.ImportModule

'Import the Surveys
'Outer Loop - To insert the Surveys
Dim intCurrentSurvey As Integer
Dim xmlSurvey As XmlNode
Dim xmlSurveys As XmlNode = GetContent(Content, "surveys")
For Each xmlSurvey In xmlSurveys
Dim SurveyInfo As New SurveyInfo
SurveyInfo.ModuleId = ModuleID
SurveyInfo.Question = xmlSurvey.Item("question").InnerText
SurveyInfo.ViewOrder = xmlSurvey.Item("vieworder").InnerText
SurveyInfo.CreatedByUser = xmlSurvey.Item("createdbyuser").InnerText
SurveyInfo.CreatedDate = xmlSurvey.Item("createddate").InnerText
SurveyInfo.OptionType = xmlSurvey.Item("optiontype").InnerText
'Add the Survey to the database
intCurrentSurvey = AddSurvey(SurveyInfo)

'Inner Loop - To insert the Survey Options
Dim xmlSurveyOption As XmlNode
```

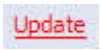
```
Dim xmlSurveyOptions As XmlNode = xmlSurvey.SelectSingleNode("surveyoptions")
For Each xmlSurveyOption In xmlSurveyOptions
Dim SurveyOptionInfo As New SurveyOptionInfo
SurveyOptionInfo.SurveyId = intCurrentSurvey
SurveyOptionInfo.OptionName = xmlSurveyOption.Item("optionname").InnerText
SurveyOptionInfo.IsCorrect = xmlSurveyOption.Item("iscorrect").InnerText
SurveyOptionInfo.ViewOrder = xmlSurveyOption.Item("vieworder").InnerText
'Add the Survey to the database
SurveyOptionController.AddSurveyOption(SurveyOptionInfo)
Next

Next ' Retrieve the next Survey

End Sub
```

C#:

```
void DotNetNuke.Entities.Modules.IPortable.ImportModule(int ModuleID, string Content, string Version,
int UserID)
{
//Import the Surveys
//Outer Loop - To insert the Surveys
int intCurrentSurvey;
XmlNode xmlSurveys = DotNetNuke.Common.Globals.GetContent(Content, "surveys");
foreach (XmlNode xmlSurvey in xmlSurveys)
{
SurveyInfo SurveyInfo = new SurveyInfo();
SurveyInfo.ModuleId = ModuleID;
SurveyInfo.Question = xmlSurvey["question"].InnerText;
SurveyInfo.ViewOrder = Convert.ToInt32(xmlSurvey["vieworder"].InnerText);
SurveyInfo.CreatedByUser = Convert.ToInt32(xmlSurvey["createdbyuser"].InnerText);
SurveyInfo.CreatedDate = Convert.ToDateTime(xmlSurvey["createddate"].InnerText);
SurveyInfo.OptionType = xmlSurvey["optiontype"].InnerText;

// Add the Survey to the database
intCurrentSurvey = AddSurvey(SurveyInfo);
//Inner Loop - To insert the Survey Options
XmlNode xmlSurveyOptions = xmlSurvey.SelectSingleNode("surveyoptions");
foreach (XmlNode xmlSurveyOption in xmlSurveyOptions)
{
SurveyOptionInfo SurveyOptionInfo = new SurveyOptionInfo();
SurveyOptionInfo.SurveyId = intCurrentSurvey;
SurveyOptionInfo.OptionName = xmlSurveyOption["optionname"].InnerText;
SurveyOptionInfo.IsCorrect = Convert.ToBoolean(xmlSurveyOption["iscorrect"].InnerText);
SurveyOptionInfo.ViewOrder = Convert.ToInt32(xmlSurveyOption["vieworder"].InnerText);

// Add the Survey to the database
SurveyOptionController.AddSurveyOption(SurveyOptionInfo);
```

```
}
}
}
```

Now, return to the module definition you created and click the Update link.



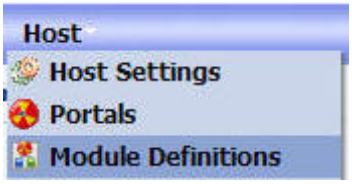
The Portable feature will now have check box.

Packaging and protecting your Module

You want to create a DotnetNuke module and you want to distribute it but you don't want to give away your source code?

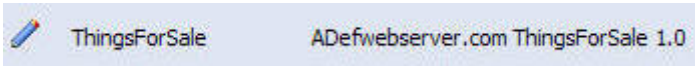
This chapter will guide you through packageing your module. This package will include the source code. The source code will be replaced with compiled assemblies (.dll's) in a later step.

While logged in as the **Host** account, select ***Module Definitions*** from the *Host* menu.



Click the *pencil* icon next to the module to select it and navigate to the details.

In this example we are using a module called **ThingsForSale** that was created in the tutorial located at http://www.adefwebserver.com/DotNetNukeHELP**.**



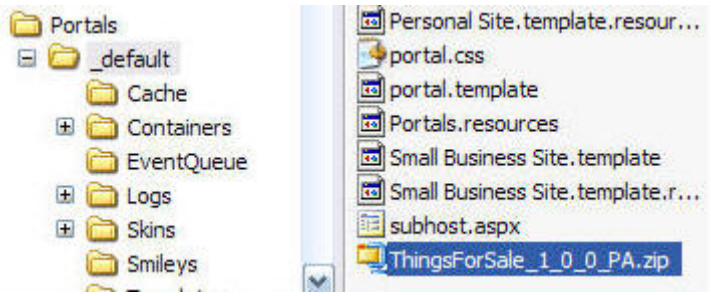
From the menu for the module definition for **ThingsForSale**, select ***Create Private Assembly***.

Enter a name that ends with *.zip* and click the *Create* link.

The Private Assembly will be created.

However it will not contain any assemblies at this point. It will contain source code.

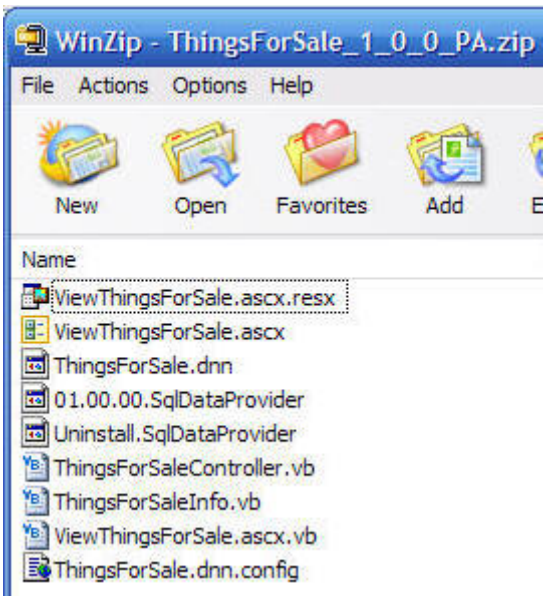
You can retrieve the *.zip* file from the root directory of the portal.



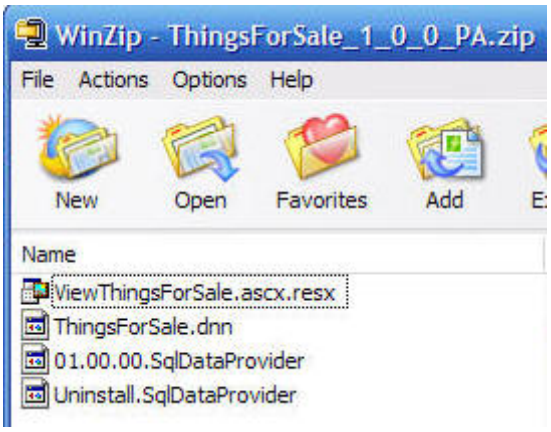
When you open the *.zip* file you can see the module elements.

The source code is contained in the files with the *.vb* extension. These will be replaced with compiled assemblies.

Note: *At this point the* ***ThingsForSale_1_0_0_PA.zip*** *file can be used to install the module. Therefore, you will want to save a copy for use when you want to distribute a source code version of your module or to archive the version of the module.*



Remove all the files with the *.vb* *and* *.ascx* and *.config* extensions from your *.zip* file



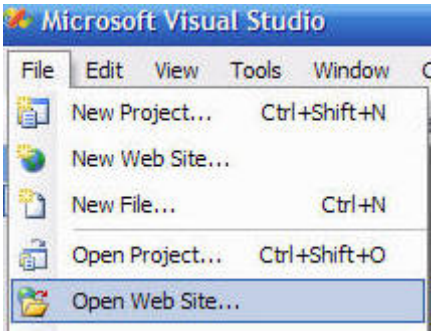
Protect Your Source Code

To create a .zip file with compiled assemblies (so that you will have a version that does not contain your source code), follow these steps:

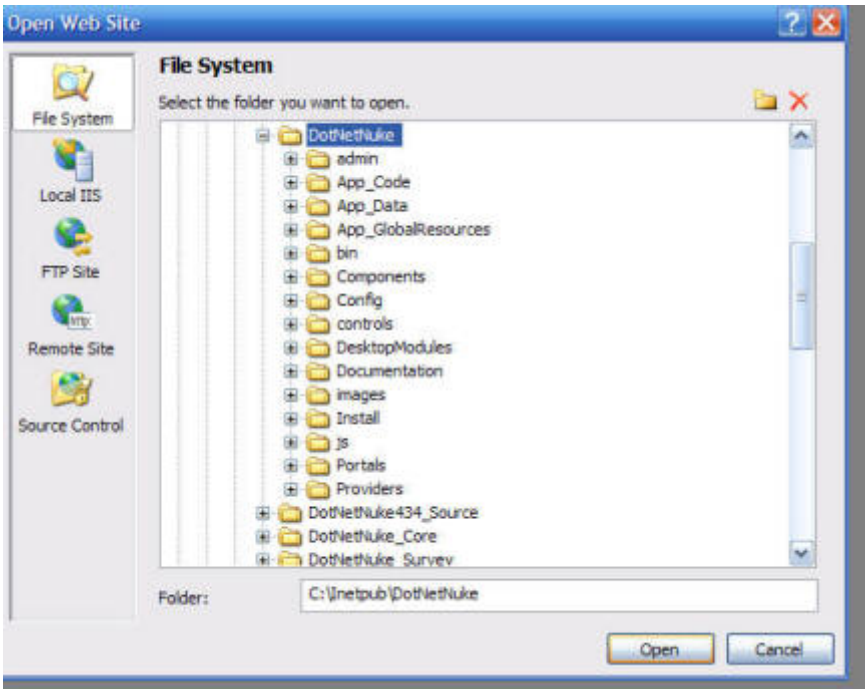
- Create the compiled assemblies (*.dll's*)
- Update the *.dnn* configuration file

(Note: You must use Visual Studio for this. you cannot use Visual Web Developer Express)

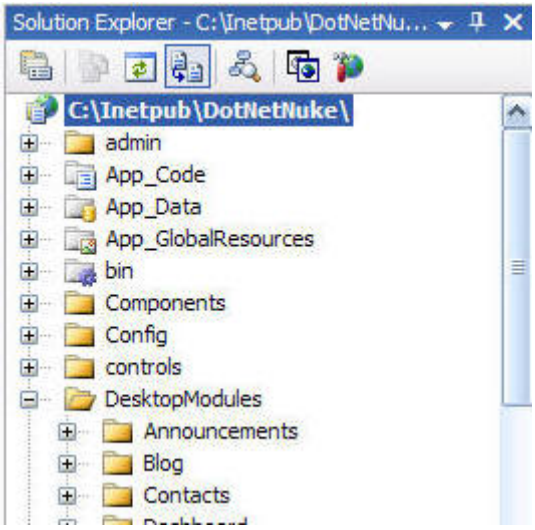
Open Visual Studio 2005 and from the **File** menu select *Open Web Site.*



Navigate to your DotNetNuke site...

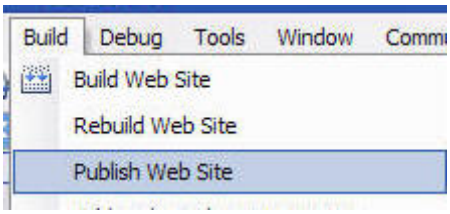


And open it



Create the compiled assemblies

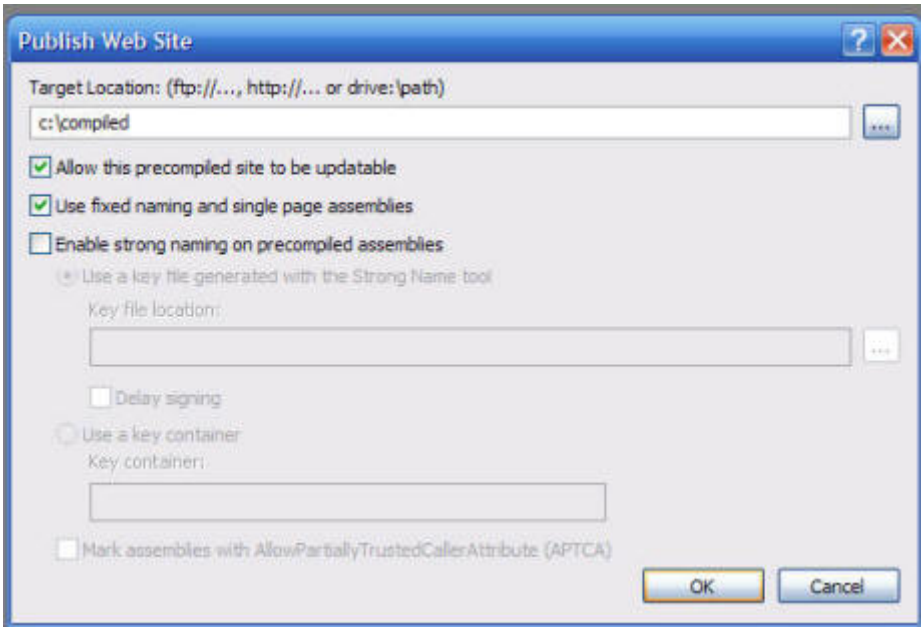
From the **Build** menu, select *Publish Web Site*



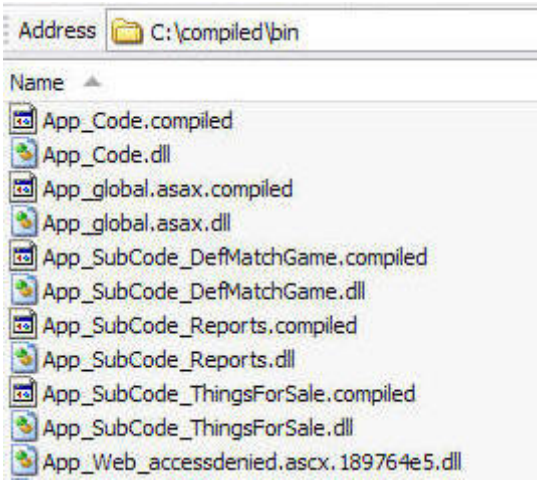
From the Publish Web Site menu, enter a path for the compiled site (for example *C:\compiled*) and check the boxes next to:

- Allow this precompiled site to be updatable
- Use fixed naming and single page assemblies

Click the **Ok** button.

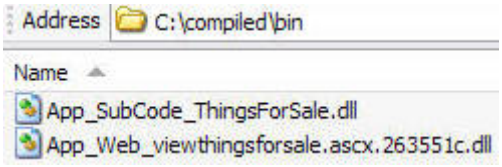


When you look in the *C:\compiled\bin* folder you can see all the assemblies.

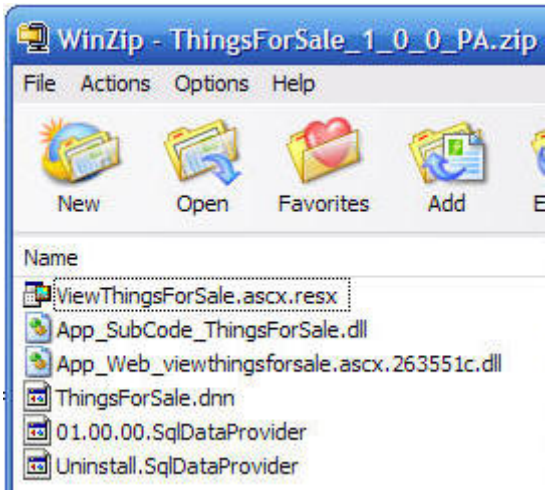


Locate any assemblies with the *.dll* extension that contain the name of the folder your module was in.

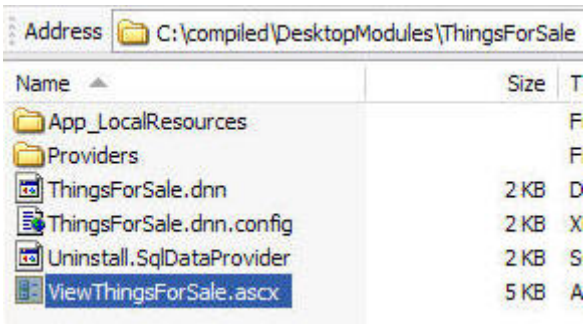
For example, the **ThingsForSale** module was in the *\DesktopModules\ThingsForSale* directory. Therefore, any assembly that has the words *thingsforsale* in its name is a compiled assembly for the module.



Place them in into your *.zip* file

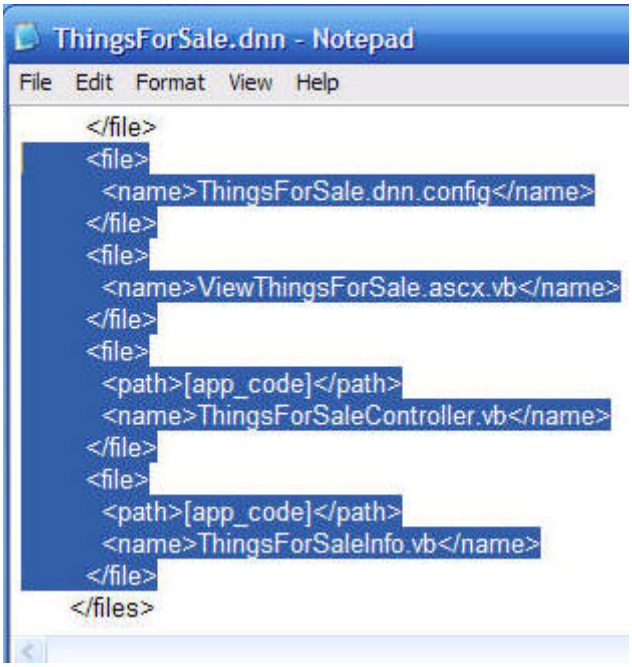


Copy any files with the *.ascx* extension that reside in the folder for your module (In this example *C:\compiled\DesktopModules\ThingsForSale*) to your *.zip* file

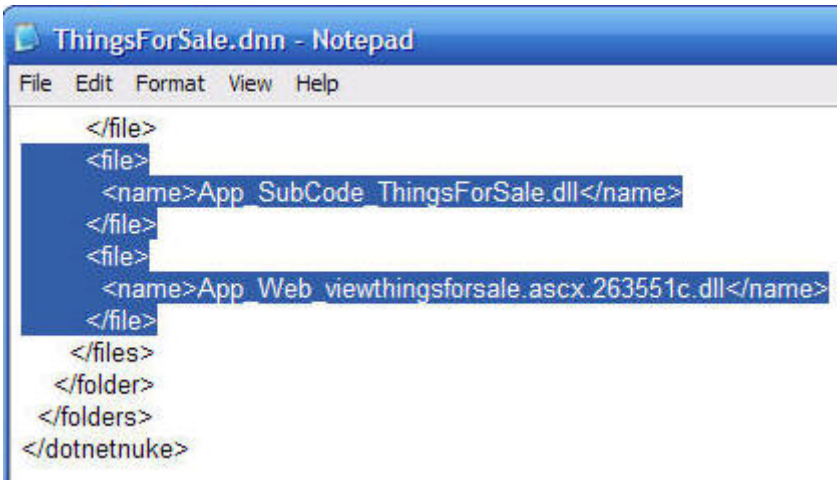


Update the *.dnn* configuration file

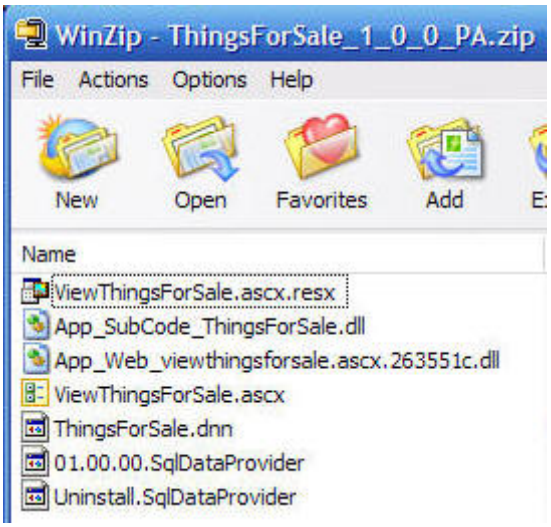
Open the *.dnn* file and remove any reference to any files with a *.vb* or a *.config* extension



- Add the names of any of the assemblies you added to the *.zip* file.
- Surround the names with *the* tags



You will now have a *.zip* file that can be installed in a DotNetNuke site.



Upgrading You Module

When you create a new version of your module you will want to allow it to be upgraded.

To upgrade the module the portal administrator will upload the new version of the module.

The DotNetNuke framework will detect that a new version has been uploaded and upgrade it.

Removing old Assemblies

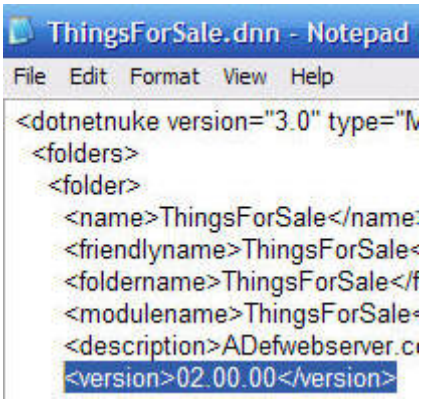
The problem you will have is the old assemblies will need to be removed, otherwise you will have a potential namespace conflict because the old and the new assemblies will both reside in the */bin* directory at the same time.

To resolve this you will need to include a *.txt* file in your *.zip* file that will instruct the DotNetNuke framework to remove **all** previous assemblies for all previous versions of the module.

Implementing the Upgrade

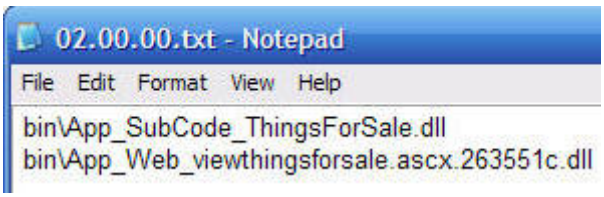
*This is assuming you have created a new version of the **ThingsForSale** module.*

The first step is to change the version in the *.dnn* file.



Next, create a text file named *02.00.00.txt* and place the names of the assemblies for all previous version of the module using ***bin/assembly name*** format.

Place this file in the *.zip* file for the new version of the module.

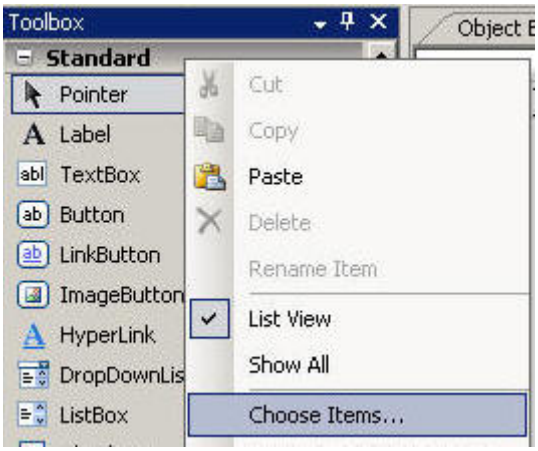


Using the DNN Web Controls

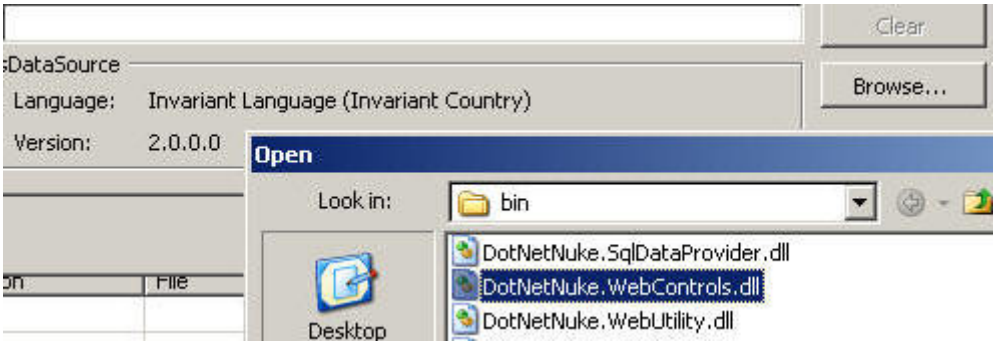
DotNetNuke comes with many user controls to aid in your module development. Here we cover three popular controls.

Set up the Web Controls

Right-Click on the Toolbox in Visual Studio and select **Choose Items...**



Use the Browse button to browse to DotNetNuke.WebControls.dll (in the /bin folder of your DotNetNuke installation) and click the OK button.



The DotNetNuke web controls will now appear in your toolbox:

The DNN Label Edit Control

To see the normal functionality of the DNN Label Edit control, click on it while logged in as the portal Administrator:

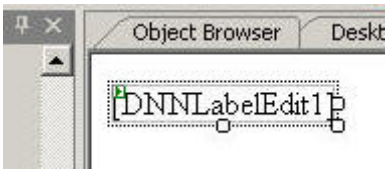
Make your changes:

And click away from the label to save your changes:

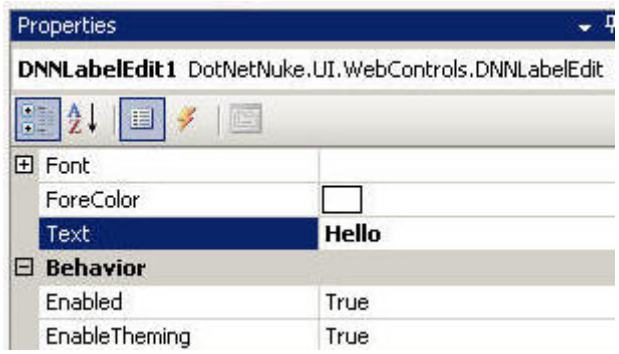
Note: The method to update this control may change in future versions but the functionality will be the same.

Using the DNN Label Edit Control

Drag the DNNLabelEdit control from the toolbox and drop it on to your DotNetNuke User Control.



In the properties for the control, set the Text property to "Hello"



Save the page and view it in the web browser

Click on the word "Hello" and type "World!" at the end.

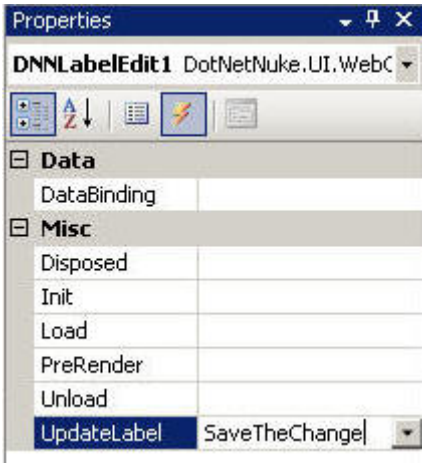
Click away from the label and the label has been updated

Saving the Changes

There is a method that is exposed by the DotNetNuke API that allows you to save module settings that are specific to that instance of the module (meaning, each time you place the module in the portal the settings for each instance is kept separate).

In the properties for the DNNLabelEdit control, click on the yellow lazerbolt to show the events for the control

Type "SaveTheChange" for the UpdateLabel event and click away from the box.



Visual Studio switches to code view and your method is all ready for your custom code.

```
Protected Sub SaveTheChange(ByVal source As Object, ByVal e As DotNetNuke.UI.W

End Sub
```

Ensure that your Web User Control has this line toward the top:

Inherits Entities.Modules.PortalModuleBase

Enter this code in the method so it appears like this:

```
Protected Sub SaveTheChange(ByVal source As Object, ByVal e As _
DotNetNuke.UI.WebControls.DNNLabelEditEventArgs) Handles
DNNLabelEdit1.UpdateLabel

    Dim moduleCtrl As New DotNetNuke.Entities.Modules.ModuleController

    Dim TheNewText As String
    TheNewText = e.Text.ToString()

    moduleCtrl.UpdateModuleSetting(ModuleId, "helloText", TheNewText)

End Sub
```

Alter the Page_Load method (or create one if you don't already have one) to this:

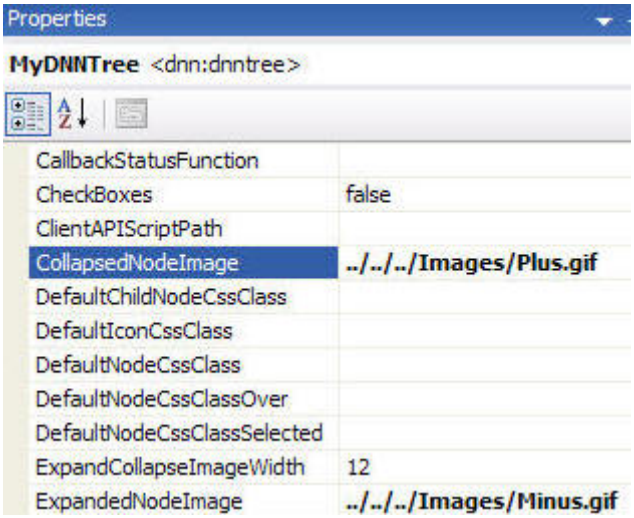
```
Protected Sub Page_Load(ByVal sender As System.Object, ByVal e As _

System.EventArgs) Handles MyBase.Load
    'If Not Page.IsPostBack Then
    Dim helloText As String = CType(Settings("helloText"), String)
    If (helloText Is Nothing) Then
        helloText = "Hello World"
    End If

    Me.DNNLabelEdit1.Text = helloText
    'End If

End Sub
```

The DNNTree Control

The **DNNTree** control will display items in an expandable tree.

Drag the DNNTree control from the toolbox and drop it on to your DotNetNuke User Control. In the properties for the control, change the ID property to **"MyDNNTree"**.

Under the properties for the control, change set the **CollapsedNodeImage** property to **"../../../Images/Plus.gif"** and the **ExpandedNodeImage** property to **"../../../Images/Minus.gif"**



When you look at the page in source view the code should look like this:

```
<dnn:dnntree id="MyDNNTree" runat="server"
collapsednodeimage="../../../Images/Plus.gif"
expandednodeimage="../../../Images/Minus.gif">
</dnn:dnntree>
```

In the code behind, ensure that your User Control has this line toward the top:

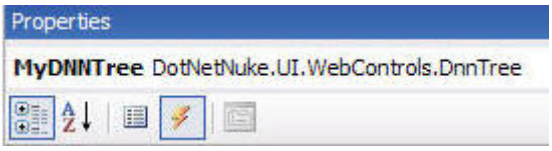
Inherits Entities.Modules.PortalModuleBase

Next, add the following code:

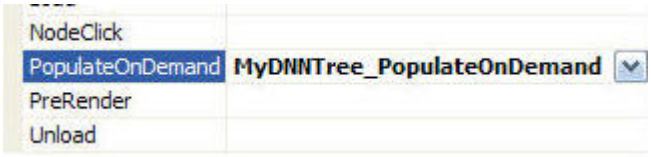
```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
If Not Page.IsPostBack Then
MyDNNTree.ImageList.Add("../../../Images/folder.gif")
MyDNNTree.ImageList.Add("../../../Images/file.gif")
PopulateTree()
End If
End Sub

Private Sub PopulateTree()
MyDNNTree.TreeNodes.Clear()
Dim index As Integer = 0
Dim objNode As TreeNode = New TreeNode("Tutorials")
objNode.NavigateUrl = "http://www.adefwebserver.com/DotNetNukeHELP/"
objNode.ToolTip = "DotNetNuke Tutorial Series"
objNode.ImageIndex = eImageType.Folder
objNode.ClickAction = eClickAction.Navigate
objNode.HasNodes = True
MyDNNTree.TreeNodes.Add(objNode)
index = objNode.TreeNodes.Add()
objNode = objNode.TreeNodes(index)
objNode.Text = "DotNetNuke 4"
objNode.ToolTip = "DotNetNuke 4 Tutorials"
objNode.ImageIndex = eImageType.Folder
objNode.ClickAction = eClickAction.Expand
PopulateChildrenTreeNodes(objNode)
End Sub
```

```
Private Sub PopulateChildrenTreeNodes(ByVal objParent As TreeNode)
Dim index As Integer = 0
Dim objTreeNode As TreeNode
index = objParent.TreeNodes.Add()
objTreeNode = objParent.TreeNodes(index)
objTreeNode.Text = "Super-Simple Module (DAL+)"
objTreeNode.NavigateUrl =
"http://www.adefwebserver.com/DotNetNukeHELP/DNN_ShowMeThePages/"
objTreeNode.ImageIndex = eImageType.Page
objTreeNode.ClickAction = eClickAction.Navigate
index = objParent.TreeNodes.Add()
objTreeNode = objParent.TreeNodes(index)
index += 1
objTreeNode.Text = "Super-Fast Super-Easy Module (DAL+)"
objTreeNode.NavigateUrl =
"http://www.adefwebserver.com/DotNetNukeHELP/DNN_Things4Sale/"
objTreeNode.ImageIndex = eImageType.Page
objTreeNode.ClickAction = eClickAction.Navigate
index = objParent.TreeNodes.Add()
objTreeNode = objParent.TreeNodes(index)
index += 1
objTreeNode.Text = "Create a full complete Module"
objTreeNode.NavigateUrl = "http://www.adefwebserver.com/DotNetNukeHELP/DNN_Module4/"
objTreeNode.ImageIndex = eImageType.Page
objTreeNode.ClickAction = eClickAction.Navigate
End Sub

Public Enum eImageType
Folder
Page
End Enum
```

Switch back to design view and click on the DNNTree control. In the properties for the DNNTree control, click on the yellow lazerbolt to show the events for the control



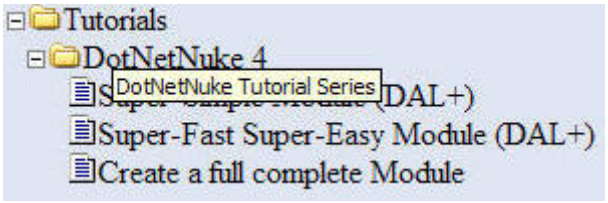
We see that there are events that we can wireup a method to. Type **"MyDNNTree_PopulateOnDemand"** for the **PopulateOnDemand** event and click away from the box.



Visual Studio switches to code view and your method is ready for your custom code. Add **"PopulateChildrenTreeNodes(e.Node)"** to the method so it appears like this:

```
Protected Sub MyDNNTree_PopulateOnDemand(ByVal source As Object, ByVal e As
DotNetNuke.UI.WebControls.DNNTreeEventArgs) Handles
MyDNNTree.PopulateOnDemand
PopulateChildrenTreeNodes(e.Node)
End Sub
```

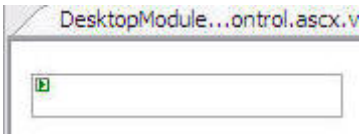
Save the page and **Build** the page. Notice that the tool tips appear when you hover your mouse over the folders.



The DNN Text Suggest Control

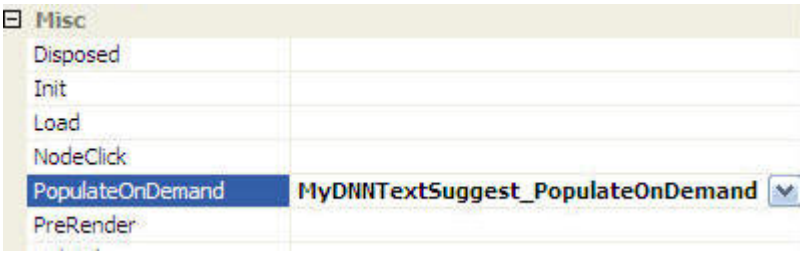
The DNNTextSuggest control allows you to type in text and provides matching items from a data source (for example a database). This provides the simular functionality as *Google Suggests* on Google.com.

Drag the DNNTextSuggest control from the toolbox and drop it on to your DotNetNuke User Control.



In the properties for the DNNTextSuggest control, click on the yellow lazerbolt to show the events for the control

We see that there are events that we can wireup a method to. Type **"MyDNNTextSuggest_PopulateOnDemand"** for the **PopulateOnDemand** event and click away from the box.



Visual Studio switches to code view and your method is all ready for your custom code. Add **"PopulateList(e.Nodes, e.Text)"** to the method so it appears like this:

```
Protected Sub MyDNNTextSuggest_PopulateOnDemand(ByVal source As Object,
ByVal e As DotNetNuke.UI.WebControls.DNNTextSuggestEventArgs) Handles
MyDNNTextSuggest.PopulateOnDemand
    PopulateList(e.Nodes, e.Text)
End Sub
```

Ensure that your Web User Control has this line toward the top:

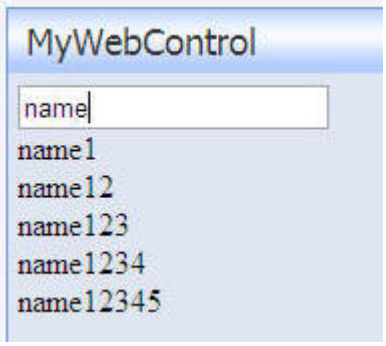
Inherits Entities.Modules.PortalModuleBase

Enter this additional code:

```
Private Sub PopulateList(ByVal objNodes As DNNNodeCollection, ByVal strText As
String)

Dim o As DNNNode
Dim dt As DataTable = GetData()
'strip out troublesome chars for Select below
strText = strText.Replace("[", "").Replace("]", "").Replace("'", "''")
dt.CaseSensitive = Me.MyDNNTextSuggest.CaseSensitive
Dim drs() As DataRow = dt.Select("name like '" & strText & "%'")
Dim dr As DataRow
For Each dr In drs
If Me.MyDNNTextSuggest.MaxSuggestRows = 0 _
OrElse objNodes.Count < (Me.MyDNNTextSuggest.MaxSuggestRows + 1) Then
o = New DNNNode(CStr(dr("name")))
o.ID = CStr(dr("id"))
objNodes.Add(o)
End If
Next

End Sub

Private Function GetData() As DataTable

Dim dt As DataTable = New DataTable
dt.Columns.Add(New DataColumn("id", GetType(String)))
```

```
dt.Columns.Add(New DataColumn("name", GetType(String)))
AddRow(1, "name1", dt)
AddRow(2, "name12", dt)
AddRow(3, "name123", dt)
AddRow(4, "name1234", dt)
AddRow(5, "name12345", dt)
Return dt

End Function


Private Sub AddRow(ByVal ID As Integer, ByVal Name As String, ByVal dt As
DataTable)
Dim dr As DataRow = dt.NewRow()
dr("id") = ID
dr("name") = Name
dt.Rows.Add(dr)

End Sub
```

Save the page and Build the page. When you type "name" in the text box it will show you matching items.



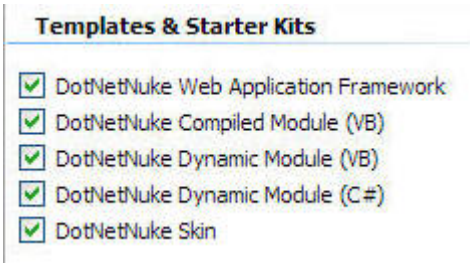
Using Web Application Projects (WAP)

What You Need:

- Visual Studio 2005 (with Service Pac 1) (This will not work with Visual Web Developer Express)
- SQL Server Express or SQL Server 2000/2005
- The latest copy of the DotNetNuke "Source Version"
- DotNetNuke Starter Kit

Install the *DotNetNuke Starter Kit*

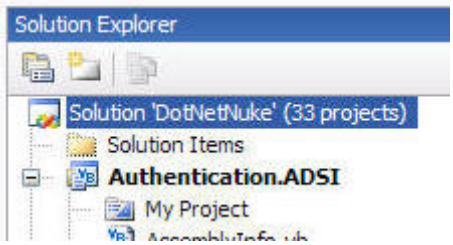
The **DotNetNuke Web Application Framework** and **DotNetNuke Compiled Module** will install as part of the *Starter Kit*



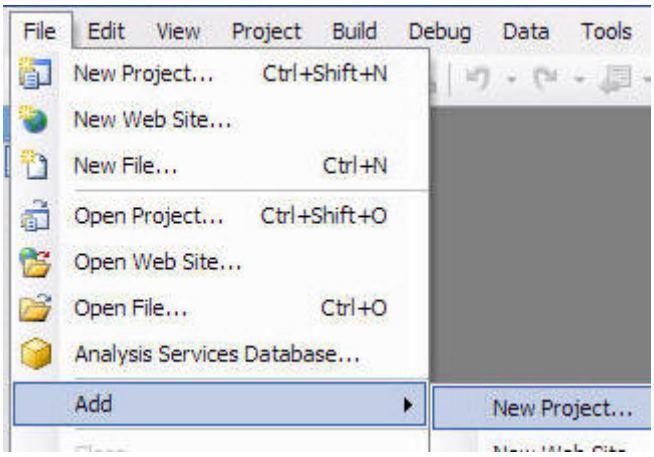
Install the DotNetNuke *Source Version*

Double-click on the *DotNetNuke.sln* file in the Windows file manager to open the Solution in Visual Studio

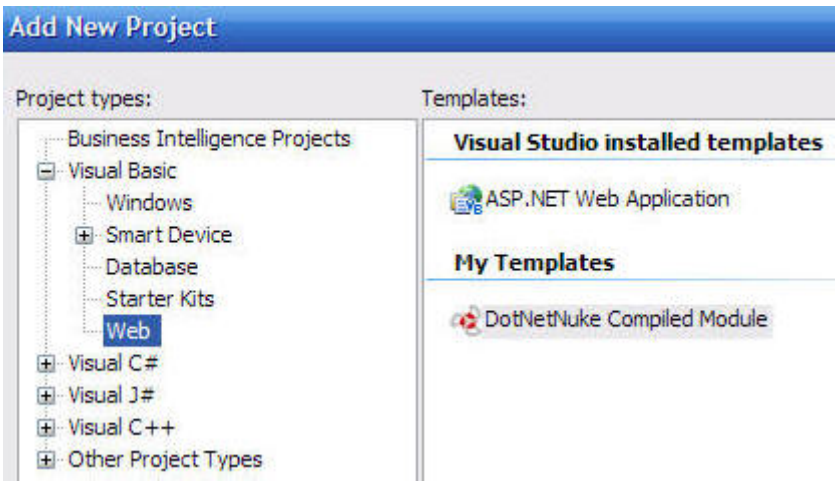
Click on the root Solution node in the the **Solution Explorer**



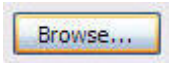
From the menu bar select **File**, then **Add**, then **New Project**



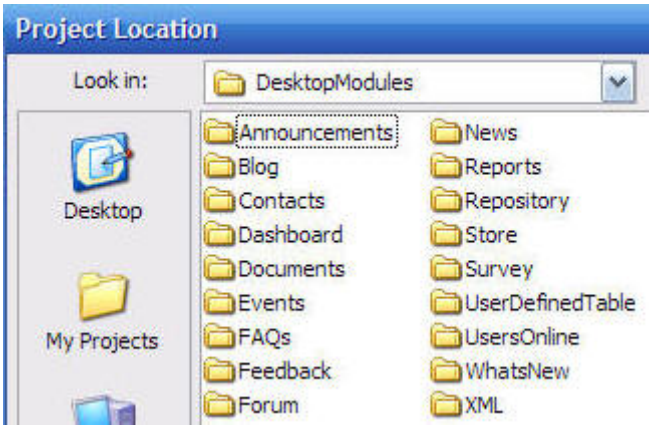
In the **Add New Project** dialog, click on **Visual Basic** then **Web** then on **DotNetNuke Compiled Module** under *My Templates*



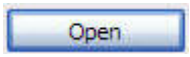
Also, in the **Add New Project** dialog, click the **Browse** button.



Navigate to the **DesktopModules** directory



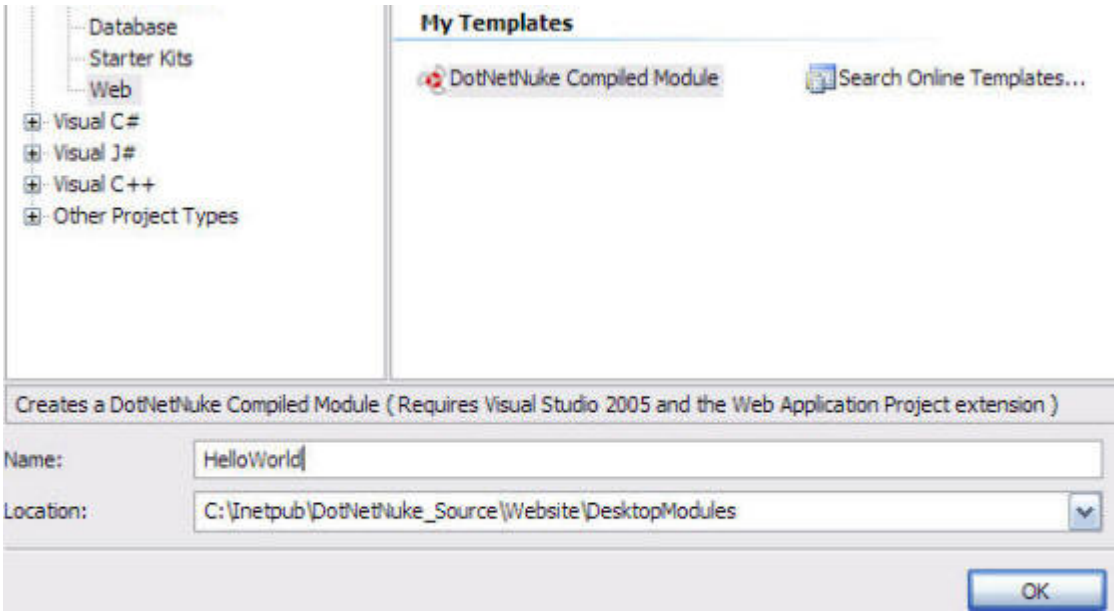
Click the **Open** button



Also, in the **Add New Project** dialog, enter *HelloWorld* in the Name box

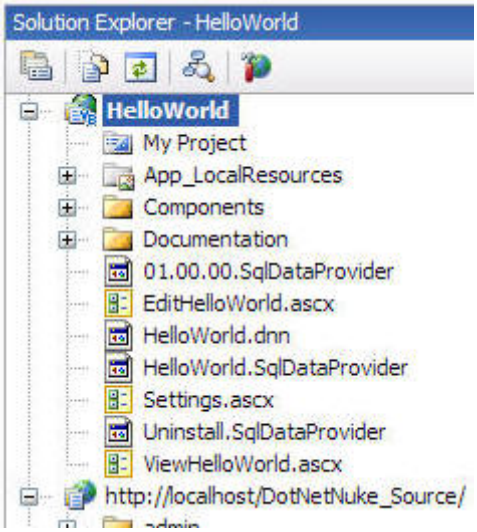
The **Add New Project** dialog should now be configured as follows:

- **DotNetNuke Compiled Module** template is selected
- **Name** is set to *HelloWorld*
- Locations is set to the **DesktopModules** directory



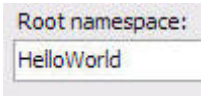
Click the **Ok** button.

The *HelloWorld* project will be created

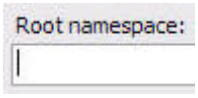


Double-click on the **My Project** icon under the *HelloWorld* project

Change the Root namespace...



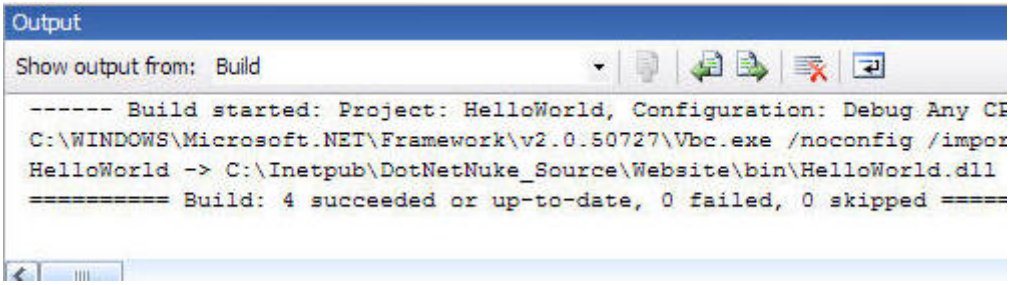
...to nothing and then click the Save icon on the toolbar.



Right-click on the project...

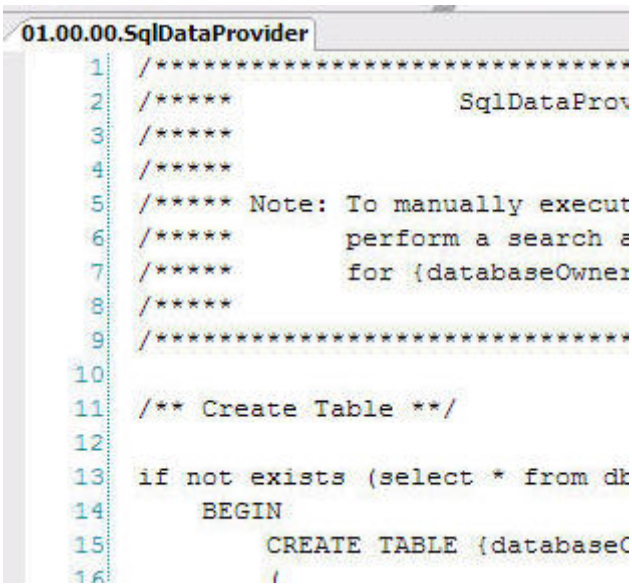
...and **Build** the Project

The *HelloWorld* project will Build



Double-click on the **01.00.00.SqlDataProvider** file under the *HelloWorld* project

Copy the entire SQL Script:

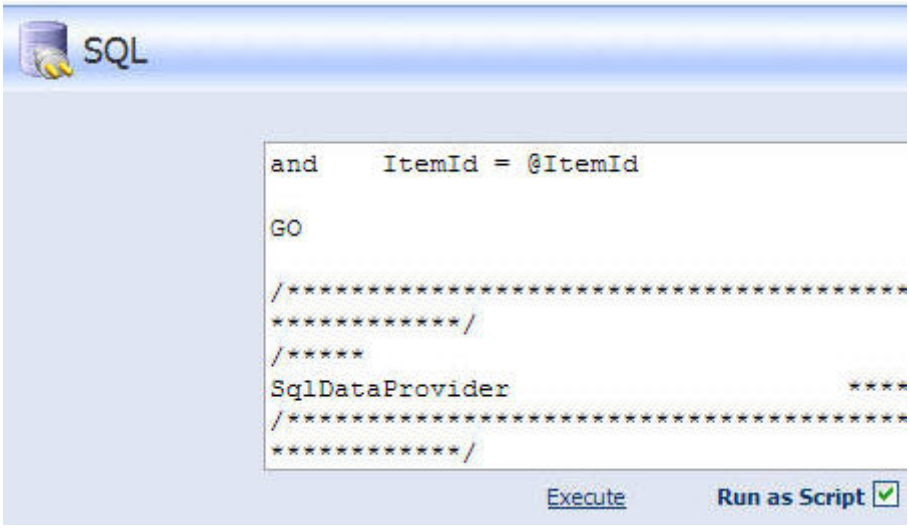


Open the site in your web browser and log in using the *Host* account.

On the **Host** menu select **SQL**

Paste the SQL script in the SQL box.

Select the **Run as Script** box and click "**Execute**".



On the **Host** menu select **Module Definitions**

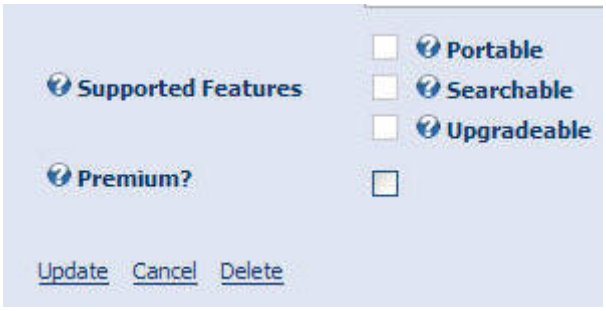
On the **Module Definitions** menu, select **Create New Module**

At the top of the **Edit Module Definitions** page, select *HelloWorld.dnn* from the **Module Manifest** *drop-down* and click the Install link

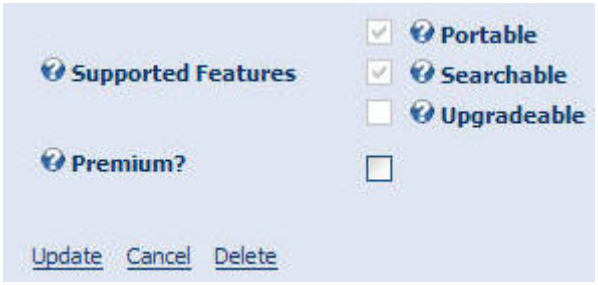
On the **Module Definitions** page click the pencil icon next to *HelloWorld* to edit the module definition

The optional interfaces (*Portable* and *Searchable*) will not be checked.

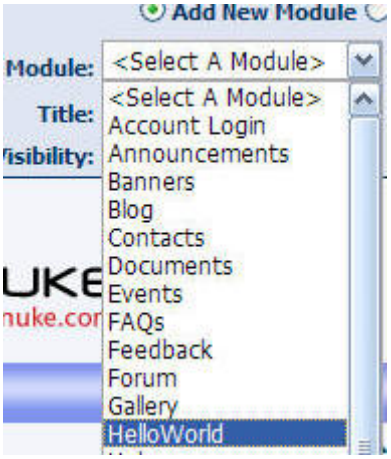
Click the **Update** Link

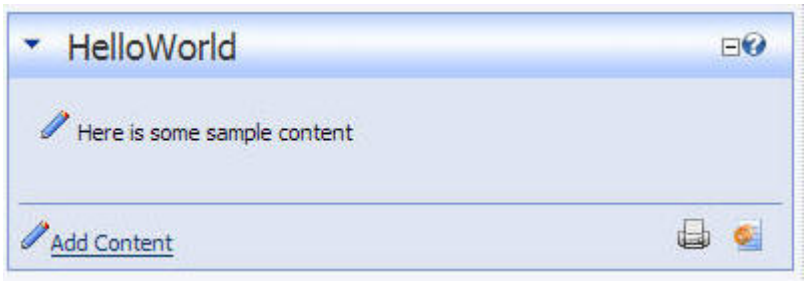


The Module Configuration will now be properly updated



The *HelloWorld* module will appear in the Module list so that you can now add it to a page.





Survey Module Source Code

The source code for the Survey module is listed in *DotNetNuke 4.0 Module Developers Guide (Part 2)* **available at DotNetNuke.com**

Additional Information

The DotNetNuke Portal Application Framework is constantly being revised and improved. To ensure that you have the most recent version of the software and this document, please visit the DotNetNuke website at:

http://www.dotnetnuke.com

The following additional websites provide helpful information about technologies and concepts related to DotNetNuke:

DotNetNuke Community Forums
http://forums.asp.net/90/ShowForum.aspx

Microsoft® ASP.Net
http://www.asp.net

Open Source
http://www.opensource.org/

W3C Cascading Style Sheets, level 1
http://www.w3.org/TR/CSS1

Errors and Omissions

If you discover any errors or omissions in this document, please email marketing@dotnetnuke.com. Please provide the title of the document, the page number of the error and the corrected content along with any additional information that will help us in correcting the error.

Appendix A: Document History

Version	Last Update	Author(s)	Changes
1.0.0	Nov 1, 2006	Michael Washington	First Draft