DotNetNuke Localization

Dan Caron, Charles Nurse, Shaun Walker

Version 1.0.11
Last Updated: June 20, 2006
Category: Localization

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.

Abstract

In order to clarify the intellectual property license granted with contributions of software from any person or entity (the "Contributor"), Perpetual Motion Interactive Systems Inc. must have a Contributor License Agreement on file that has been signed by the Contributor.

Contents

DotNetNuke Localization

Introduction

DotNetNuke is used internationally by web developers to quickly build professional web applications. Given the International community that DNN serves, it is necessary to allow DNN to build systems that cater to the specific culture of both the hosting organization, and the individual end users/site visitors. In some instances, these sites will serve a wide range of cultures while other sites target a specific culture.

Localization Scope

There are a number of different areas to consider when it comes to localization.

- **Static Resources** - these are the text labels, messages, and graphics that are spread throughout the user interface of the application. In the case of DotNetNuke they are currently hardcoded into the various user control (.ascx) files with some dynamic message specification in the code behind (.ascx.vb)
- **Dynamic Content** – for a site to be completely localized, there should be a way for the content of the site to be managed in multiple languages. Since automatic translation is not reliable there must be a way to store multiple versions of content for each language. This functionality borders on the features provided by Content Management Systems (CMS) which is a large topic unto itself and will not be discussed in this document.
- **Dates, Numbers, Currencies** – each culture has its own regional format for these types of information. This item impacts both static resources and content.

Localization Components

In order to implement any type of localization system there are a number of key components which need to be implemented. Each key component has a number of implementation options to consider.

- **Data Store** – this involves some type of storage mechanism for textual translations. It requires some type of access mechanism which allows for immediate retrieval based on a key or ID for a specific language.
- **API** – an interface is required which enables the application to retrieve the appropriate text translation from the data store at runtime. Methods must be available for both static and dynamic text translation. The API must satisfy the performance and scalability requirements of the application.

Data Store

A number of options are available for implementing a localization data store.

- **Loose Text Files** – static text or XML files could be maintained for each language. The resource file could be cached in memory for performance with a dependency placed on the file.
- **Windows Resource Files** – using the .NET Framework Localization namespace a resource file (*.resx) could be used containing binary and text information.
- **Database** – a relational database could be used to store the localization data.

API

Data Store	Pros	Cons
Loose Text Files	- simple to manage and deploy	- can only handle text data
Windows Resource Files	- can handle both text and binary data - industry standard for Windows desktop	- must be compiled as satellite assemblies - difficult to manage and deploy
Database	- centralized storage is good for web farm environments	- may result in poor performance - difficult to manage and deploy

Static Resources

The initial release of Multilingual Translations will include only static translations. This means that we will convert static strings, but will not convert "dynamic content" (i.e. content stored in the database). We will use the Windows Resource Files (RESX) format to store our translations, however we will not use the Resource Manager to access the RESX, instead they will be accessed using the System.XML namespace. We have aligned the localization solution as closely as possible to the specifications for ASP.NET 2.0.

Three Types of Translations

- **Application Resources** – These are the translations that are shared throughout many controls within DNN. For instance, to localize the words "True" and "False", you would store the translations in the Application Resource files. Other things you

might localize here are “Yes” and “No”, “On” and “Off”, “Submit”, “Continue”, “Next”, etc.

- **Local Resources** – These are the resources that are unique to a particular piece of functionality. For instance, to provide translations for the Announcements module, you would store its translations in a Local Resource location. However if the Announcements module has a static text element of “True” or “False”, there is no need to translate those in the Local Resources because it is handled already in the Application Resources. Local Resources are directly related to a User Control or Page.
- **Global Resources** – These are resources that are accessed from components that cannot be handled by Local Resources, and are not necessarily shared translations, therefore they do not fit in the categories above. Since all Local Resources are tied directly to a User Control or Page, components have no place to store their translations, therefore we have created a third category for these resources.

Four Classifications of Locales

A locale is the combination of a Language Code and a Country Code. Many languages are spoken in more than one country, and dialects may differ from one country to the next. For instance, English in the United States is slightly different from English in the United Kingdom…and French in Canada is slightly different from French in France. A locale accounts for this differentiation. Currently we support four types of locales as follows:

- **System Locale** – this is the locale that DotNetNuke is coded to by default. Our System Locale is “en-US”.
- **Default Portal Locale** – this is the portal’s default locale. Each portal can specify its default locale. This will be the locale that is automatically selected when a user (with an unknown locale) visits the portal site. The default locale can be change in the Site Settings page.
- **User Selected Locale** – this is the locale that the user selects when they visit the portal.

- **Custom Locale** – a custom locale allows us to append custom text to the locale's key to reach a level of customization in translations. Currently, we are supporting a custom locale to provide portal-level translation customization capability. So instead of the locale looking like this "en-US" it would look like this "en-US.Portal-2" which represents a custom set of translations for portal id 2. More on this later.

The System Locale, Default Portal Locale, and the User Selected Locale support the use of a Fallback Locale. A fallback locale is used when a translation for a particular locale isn't found in the translation file. Custom Locales do not support the Fallback Locale feature.

Translation Fallback

There is a hierarchy of translation files called the Translation Stack that is important to understand. When loading the translations for a control, we need to consider that a translation may not be available for some translation keys. When a translation isn't available in a particular resource file, we fallback onto another resource file to try to get the translation from there. And so on, and so forth until we have a single translation item that we cache. A single translation item in the cache will sometimes be a combination of any or all of the types of locales listed in the diagram below. It is a rollup of the following hierarchy of translations:

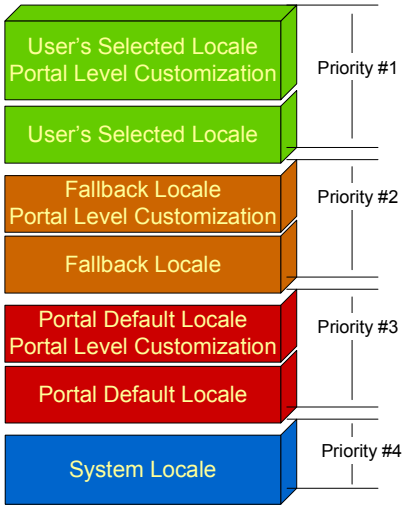
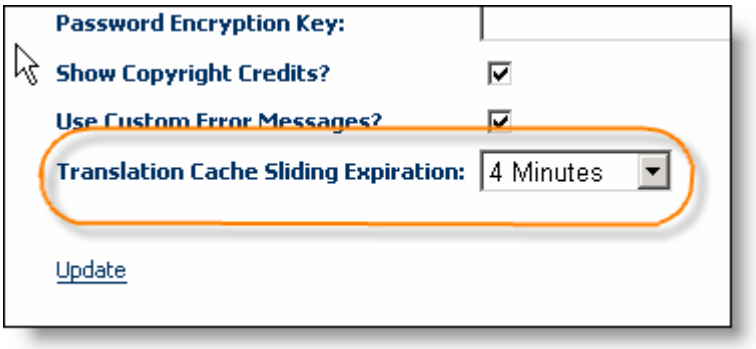


Fig. 1a – Translation Stack

Example: In Announcements.ascx, there is a hyperlink server control with the link text of "read more...". Its resource key is "ReadMore". If there are any translations in the User's Selected Locale that have been customized at the portal level, that has the highest priority and will be displayed if that key exists. If it does not exist, we check the User's Selected Locale translations to see if the key is there. If it is, we use it...otherwise we continue down the Translation Stack until we have exhausted all possibilities. The last place we look is at the System Locale level...this would be "en-US".

Translation Caching

Translations are cached on the server for a period of time that is customizable in the Host Settings tab. We are using a sliding expiration that defaults to 4 minutes. This means that if after 4 minutes a cached translation isn't accessed, it will be cleared out of the cache. After "x" minutes of inactivity on that cached translation it will be cleared from the cache. And "x" is defined in Host Settings as "Translation Cache Sliding Expiration".

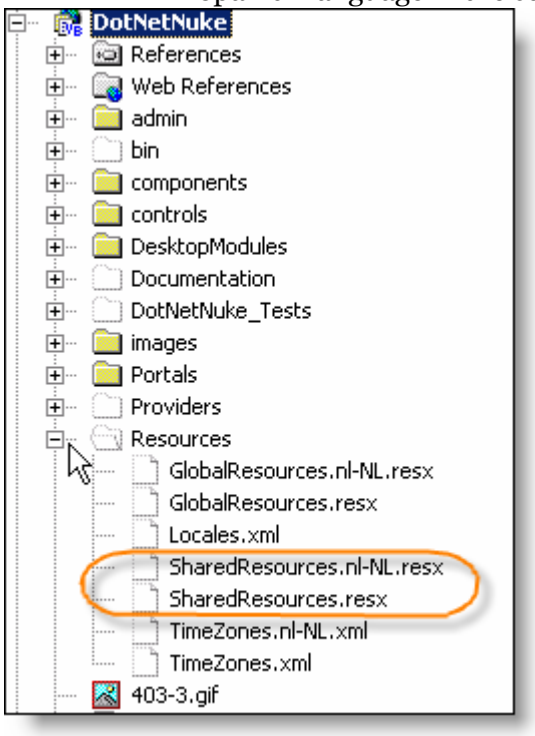


Storage of Translation Files

There are three possible areas for RESX files to be stored in:

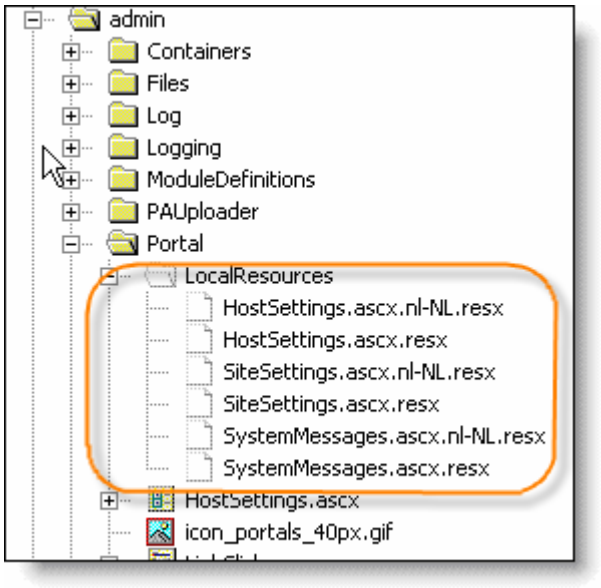
Application Resources are stored in the /Resources folder which is directly under DotNetNuke root folder.

- The file name for the System Locale is SharedResources.resx
- The file name for other locales follow this naming convention: /Resources/SharedResources.[locale].resx
    - For instance, for the Spanish language in the country of Spain, the Application Resource file would be located here: /Resources/SharedResources.es-ES.resx, where "es-ES" represents the Spanish language in the country of Spain.



Local Resources are stored in a directory within the user control's directory

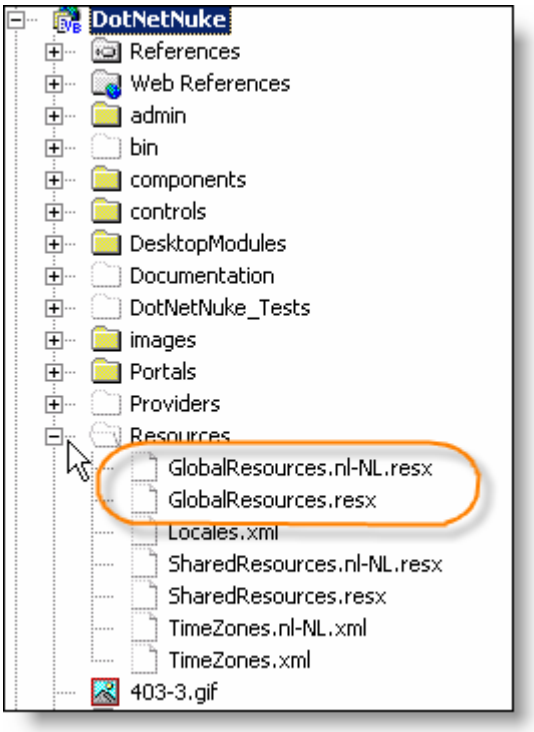
- The file name for the System Locale follows this naming convention: [control_directory]/LocalResources/[control_files_name].resx
    - For instance, Announcements/LocalResources/Announcements.ascx.resx
- The file name for other locales follows this naming convention: [control_directory]/LocalResources/[control_files_name].[locale].resx
    - For instance, Announcements/LocalResources/Announcements.ascx.es-ES.resx



- Portal-Level Customization
    - You may customize Local Resources by providing a RESX file for each portal that requires custom translations.
- The file naming convention for portal level customization is: [control_directory]/LocalResources/[control_files_name].[locale].Portal-[portal id].resx
    - For instance, Announcements/LocalResources/Announcements.ascx.es-ES.Portal-2.resx

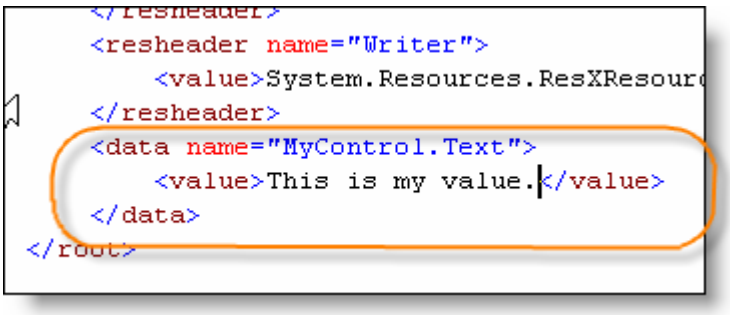
Global Resources are stored in the /Resources directory.

- The file name for the System Default locale (en-US) is: /Resources/GlobalResources.resx
- For locales other than the System Default, the naming convention is: /Resources/GlobalResources.[locale].resx.
    - For instance, /Resources/GlobalResources.es-ES.resx.
- Usage of Global Resources: In example, /admin/Containers/ActionBase.vb is a component that stores its translations in the Global Resources files because the component is not tied to any particular user control, therefore it cannot use Local Resources.



RESX File Format

To create a new RESX file, use the **/Resources/Template.resx** file as a template. Remove the section highlighted below and replace with your translation keys.



The “name” attribute value needs to match the **resourcekey** (more on this below) in the control, with “.Text” appended to it. For instance, if your resource key is “HelloWorld”, the value of the name attribute would be “HelloWorld.Text”. Make sure you pay attention to use the proper case…make sure the “T” in Text is capitalized.

RESOURCE KEYS ARE CASE SENSITIVE.

The *value* element should have its inner contents set to the actual translation, as shown above.

As shown above Resource Keys use a “name.ext” format where the default extension is “.Text”. A *resourcekey* can be defined with no extension. In this case the “.Text” extension is assumed. However, other extensions can be used. In this case the *resourcekey* attribute must explicitly use the extension. DotNetNuke uses a number of extensions which PA developers are encouraged to use.

- .Text – used for the text properties of controls (default
- .Help – used for help text (see more below)
- .Header – used for the headertext properties of DataGrid columns
- .EditText – used for the edittext properties of DataGrids
- .ErrorMessage – used for the ErrorMessage property of Validator controls

As a rule the extension should match the control property name that is being localized.

The DotNetNuke Label Control

It became apparent early on in implementing localization in the core, that a custom user control would come in handy, for four reasons:

1. Localization
2. 508 support
3. Help implementation
4. Consistency

Most Administration pages have built in 508 support. In earlier versions of DotNetNuke, this was implemented using the <label> html tag:

```
<label for="<% = txtTitle.ClientId %>">Name:</label>
<asp:textbox id="txtTitle" runat="server" />
```

In most browsers the label tag is ignored and the content is displayed normally, but browsers developed for the sight impaired would use the tag to describe what the content is used for, ie it is a label identifying what should be in the textbox named txtTitle.

Users have consistently requested more help/documentation be available in the DotNetNuke core. One method of accomplishing this would be to provide field level help attached to the labels.

The DotNetNuke LabelControl accomplishes these tasks. The html for the control is quite simple:

```
<label id=label runat="server">
  <asp:imagebutton id=imgHelp runat="server"
imageUrl="~/images/help.gif"></asp:imageButton>
  <asp:label id=lblLabel runat="server" enableviewstate="False"></asp:label>
</label>
<br>
<asp:panel id=pnlHelp runat="server" cssClass="Help">
  <asp:label id=lblHelp runat="server" enableviewstate="False"></asp:label>
</asp:panel>
```

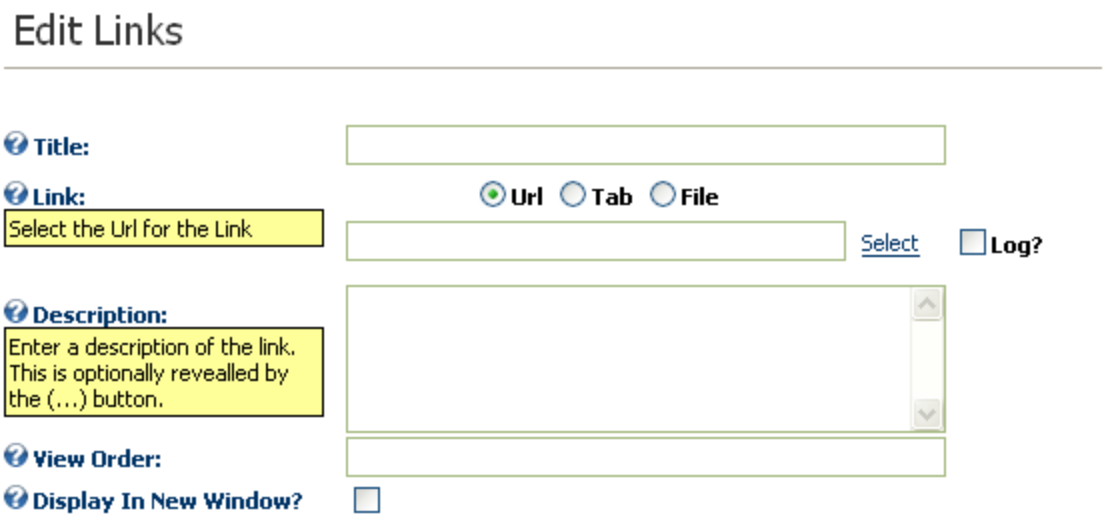
The label tag provides the 508 support, the imagebutton displays a help icon, that when clicked reveals the help text wihin the panel. The label control holds the label text. The showing and hiding of the Help is controlled using the ClientAPI (__dnn_Help_OnClick() function).

The control also exposes five properties:

- ControlName – the name of the control that this label refers to (508 support)
- ResourceKey – the resource key used for the localized label text
- HelpKey – the resource key used for the localized help text
- Suffix – an optional suffix that is added to the ResourceKey after localization
- Text – default text, if no localization text can be found.

In addition, the logic within the control also allows for localization without specifying the specific keys. If the ResourceKey is missing the control will use its ID as the key (ie ID.Text in the resx file). If the HelpKey is missing, the control will use ResourceKey.Help as the resource key for the help.

The figure below shows the control being used on the Edit Links Page. 2 labels have their help box expanded.



Converting a Portal Module to Support Translations

1. Steps for replacing Static Strings (not affiliated with an edit type control)

- Replace Static HTML Text with a server side asp:label control.
    - Static strings are any text that is displayed on the screen that is not returned from the database or XML file.
- Add a "resourcekey" attribute to the Label control, see below
    - the value of the "resourcekey" attribute should clearly represent the translation while remaining concise
        - example: If you have a module named HelloWorld.ascx which has a server-side web control with an ID of "txtFirstName", your localization key might be: resourcekey="FirstName"
- Note that if the text is an HTML label for a form field and it isn't in a <label/> tag, please add this label tag. This helps keep DNN in compliance with browsers for people with disabilities
- Also, if you notice any attributes that are missing double quotes, etc…please fix these, too.
- If a label is followed by punctuation like a colon or a comma (":" or ","), make sure the punctuation is included in the translation. Do not hard-code punctuation in the control. Punctuation differs from locale to locale. Notice in the example below the ":" after "Bar Graph Width" is inside the label. Likewise, the colon should be in the translation file as well.

Before	After
<TD class="SubHead">Title:</TD>	<TD class="SubHead"><asp:Label id="lblTitle" resourcekey="Title" runat="server" >Title:</asp:Label></TD>

2. Steps for replacing Static Text or asp:label controls that are affiliated with an edit type control (asp:textbox, asp:checkbox etc)

- Replace Static HTML Text (or existing asp:label control) with a server side DotNetNuke LabelControl.
- Optionally add a “resourcekey” and “helpkey” to the dnn:LabelControl, or alternatively you can use “ID.Text” and “ID.Help” as the keys.
- Add a “controlname” attribute to the dnn:LabelControl, to identify the control to which this label refers.
- Optionally add a text attribute for default text when no local resource file exists

Before	After
<td class="SubHead"> Bar Graph Width: </td> <td> <asp:textbox id=”txtWidth” runat=”server” /> </td>	<td class="SubHead"> <dnn:label id=dlWidth” runat=”server” controlName=”txtWidth” resourcekey=”Width” helpKey=”WidthHelp text=”Bar Graph Width:” /> </td> <td> <asp:textbox id=”txtWidth” runat=”server” /> </td>

Note: the only required attribute is the controlName to provide 508 support. If the resourcekey and helpkey were omitted, the associated resource file would look like:

```
<?xml version="1.0" encoding="utf-8" ?>
……
<root>
   <data name="dlWidth.Text">
      <value>Bar Graph Width!</value>
   </data>
   <data name=" dlWidth.Help">
      <value>Please enter the Bar Graph in the text box to the right</value>
   </data>
</root>
```

3. Convert existing labels missing the “resourcekey” attribute using the instructions above

Before	After
<asp:label id="lblPortalName" runat="server">Title</asp:label>	<asp:label id="lblPortalName" resourcekey="PortalName" runat="server">Title</asp:label>:

4. Convert existing buttons

Before	After
<asp:linkbutton id="cmdRenew" runat="server" Text="Click Here To Renew/Extend Your Portal Hosting Subscription" cssclass="CommandButton "></asp:linkbutton>	<asp:linkbutton id=cmdRenew resourcekey="Renew" runat="server" cssclass="CommandButton">Click Here To Renew/Extend Your Portal Hosting Subscription</asp:linkbutton>

5. Convert existing JavaScript from a Code-Behind file:

Before	After
cmdDelete.Attributes.Add("onClick", "javascript:return confirm('Are You Sure You Wish To Delete This Portal ?');")	cmdDelete.Attributes.Add("onClick", "javascript:return confirm('" & Localization.GetString("DeleteMessage") & "');")

6. Convert static ListItem values:

Before	After
<asp:ListItem Value="0">Private</asp:ListItem> <asp:ListItem Value="1" Selected="True">Public</asp:ListItem>	<asp:ListItem resourcekey=" PublicResults" Value="0" Selected="True">Public</asp:ListItem> <asp:ListItem resourcekey =" PrivateResults" Value="1">Private</asp:ListItem>

7. Modify or Create the RESX file to accompany the user control

- For each “key” attribute you added to a User Control, you need to have an associated value defined in the RESX that accompanies that User Control.
    - For example, if our user control looked like this before we convert it:

/DesktopModules/HelloWorld/HelloWorld.ascx - Before

```
<%@ Control language="vb" Inherits="DotNetNuke.HelloWorld"
CodeBehind="HelloWorld.ascx.vb" AutoEventWireup="false" Explicit="True" %>

<ASP:LABEL ID="lblHelloWorld" Runat="server">Hello
World</ASP:LABEL><BR>

What is your favorite color?

<ASP:DROPDOWNLIST ID="ddlQuestion" Runat="server">

   <ASP:LISTITEM Value="red">Red</ASP:LISTITEM>

   <ASP:LISTITEM Value="blue">Bluc</ASP:LISTITEM>

</ASP:DROPDOWNLIST>

<ASP:LINKBUTTON ID="btnSubmit"
Runat="server">Submit</ASP:LINKBUTTON>
```

- Then we convert it, it looks like this:

/DesktopModules/HelloWorld/HelloWorld.ascx - After

```
<%@ Control language="vb" Inherits="DotNetNuke.HelloWorld"
CodeBehind="HelloWorld.ascx.vb" AutoEventWireup="false" Explicit="True"
%>

<ASP:LABEL ID="lblHelloWorld" resourcekey="HelloWorld”
Runat="server">Hello World</ASP:LABEL><BR>

<ASP:LABEL ID="lblQuestion" resourcekey="Question" Runat="server">What
is your favorite color?</ASP:LABEL>

<ASP:DROPDOWNLIST ID="ddlAnswer" Runat="server">

   <ASP:LISTITEM resourcekey="Red" Value="red">Red</ASP:LISTITEM>

   <ASP:LISTITEM resourcekey="Blue" Value="blue">Blue</ASP:LISTITEM>

</ASP:DROPDOWNLIST>

<ASP:LINKBUTTON ID="btnSubmit" resourcekey="Submit"
Runat="server">Submit</ASP:LINKBUTTON>
```

- The Local Resource file for the locale “en-US” would look like this:

/DesktopModules/HelloWorld/LocalResources/LocalResources.ascx.en-US.resx

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <data name="HelloWorld.Text">
    <value>Hello world!</value>
  </data>
  <data name="Question.Text">
    <value>What is your favorite color?</value>
  </data>
  <data name="Red.Text">
    <value>Red</value>
  </data>
  <data name="Blue.Text">
    <value>Blue</value>
  </data>
  <data name="Submit.Text">
    <value>Submit</value>
  </data>
</root>
```

- The Local Resource file for Spanish in Spain would look something like this:

/DesktopModules/HelloWorld/LocalResources/HelloWorld.ascx.es-ES.resx

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <data name="HelloWorld.Text">
    <value>Hola Mundo!</value>
  </data>
  <data name="Question.Text">
        <value>Cuál es su color preferido?</value>
  </data>
  <data name="Red.Text">
    <value>Rojo</value>
  </data>
  <data name="Blue.Text">
    <value>Azul</value>
  </data>
  <data name="Submit.Text">
    <value>Enviar</value>
  </data>
</root>
```

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://www.dotnetnuke.com/tabid/795/Default.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	2003	Dan Caron, Charles Nurse, Shaun Walker	• Created
1.0.1	Aug 16, 2005	Shaun Walker	• Applied new template