DotNetNuke Navigation WebControls

Jon Henning

Version 1.0.0
Last Updated: June 20, 2006
Category: WebControls

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

Article I. Introduction

Early on in the creation of DotNetNuke (back then it was called the IBuySpy Portal Workshop), the need to provide a navigation control that handled the Tab (page) hierarchy was identified. Since ASP.NET 1.x had no such control that worked cross-browser (the Microsoft TreeView utilized a behavior, thus only worked well with IE), an open-source control was sought after. The author of such a control (Solution Partner's Hierarchical Menu) was contacted and soon after was integrated into the Core. Since then there have been some updates to allow for new features in the way the menu was skinned, but there really wasn't much effort put into abstracting the menu away from the core. This document is intended to outline some of this effort, in addition to offering some insight into the development of the DotNetNuke Navigation WebControls

Article II. Goals

- Provide some insight on the design decisions that went into the creation of the DotNetNuke Navigation WebControls
- Provide information on the abstraction of the menu from the core
- Provide information on how Populate On Demand functions
- Provide an overview of how the new ASP.NET 2.0 hierarchical controls will fit into DotNetNuke

Article III. DotNetNuke WebControl Design Decisions

Before diving into the navigation abstraction it is important to have a handle on some of the design decisions behind the DotNetNuke WebControls. Unlike most ASP.NET controls the DotNetNuke WebControls are designed to send the XML representing the nodes, rather than the HTML to and from the client. This has the following advantages.

- **Performance** - Server Load and Bandwidth
    - It is usually a safe assumption that the representation of the data is much smaller than the markup to display the data. Stated another way, the XML for the data will almost always be smaller than the HTML for the rendering of the data. Thus sending down the XML yields smaller output in your HTML stream.

- In many controls the maintaining of the Node's state is required. This is typically accomplished by placing the information in viewstate. Which when you break it down this is usually the same information in the data (XML). So what ends up happening in the control is it not only sends down the HTML for the data representation, it also sends down the data and the unreadable format known as viewstate. Thus, the HTML output is now twice as large.

- If your data is somewhat static, it is possible to persist the data to an xml file and reference it in a browser like IE and have the data get cached on the client (just like an image or js file). Thus, future visits to the page will not require any data transmission at all.

- **Functionality** - Being able to render your controls on the client when given nothing more than the XML data allows for many more possibilities to implement client-side functionality

  - For example, the Solution Partner's Menu Module released for DotNetNuke allows the menu to be completely customized (i.e. add/update/delete nodes, style changes, etc.) in a WYSIWYG mode. To leave the control's rendering up to the server would require round-trips to the server whenever the user wished to see the results of changing properties.

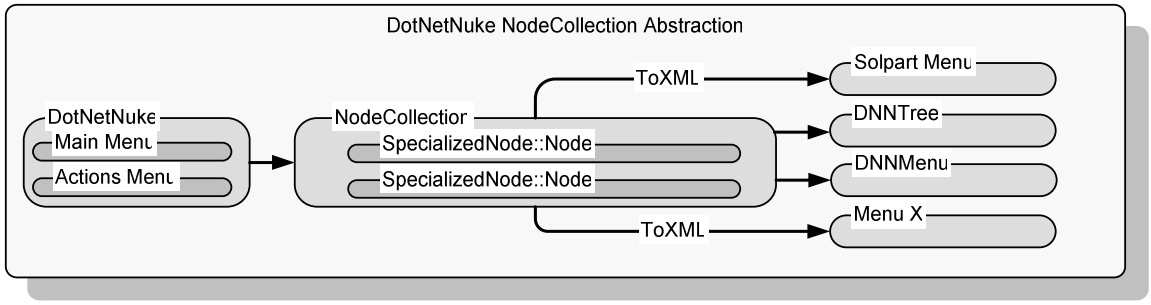
  - Enabling an on-demand loading of child nodes becomes quite simple. Simply make a client callback to the server passing the current nodeID, have the server return the XML for the child nodes, append the XML into the existing data, and re-render the control.

  - Typically controls that render on the client provide a much richer client object-model. This generally happens because the developer already spent much of the time getting things to render client-side. To add additional properties, methods, and events is not as far of a stretch.

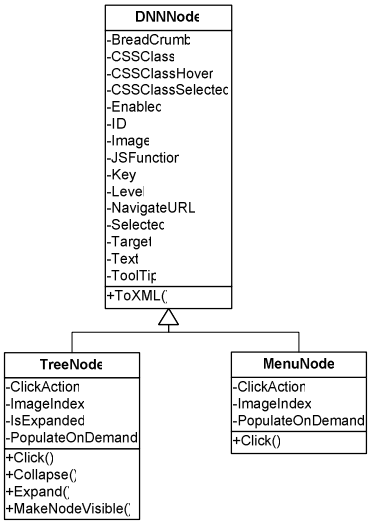
Article IV. The Need for a Generic Navigation collection

The current state of DNN (3.1 and prior) has the building of the page navigation done in each user control (SolpartMenu.ascx, TreeViewMenu.ascx). This is accomplished by looping through the tabs collection and adding the menu items/treenodes based on the current user's permissions. In order to centralize this logic a new class called Navigation was created that will expose the navigation hierarchy in a new node collection called DNNNodeCollection. This collection is really nothing more than a

wrapper around a XMLDocument and XMLNode objects, therefore it can be easily converted to XML and allows complex XPath queries inherently.



As the diagram shows, the DotNetNuke controls can digest the collection directly, whereas the third party controls like the Solpart menu are given an XML string to parse. You may be wondering how it is possible that different controls like a tree and a menu can consume the same collection directly. This is possible because the specialized nodes for both the DNNTree and the DNNMenu inherit from the DNNNode class. This class contains the fundamental properties and methods that all navigation nodes should have. The following diagram outlines the main properties for both the specialized and generic DNNNode objects.

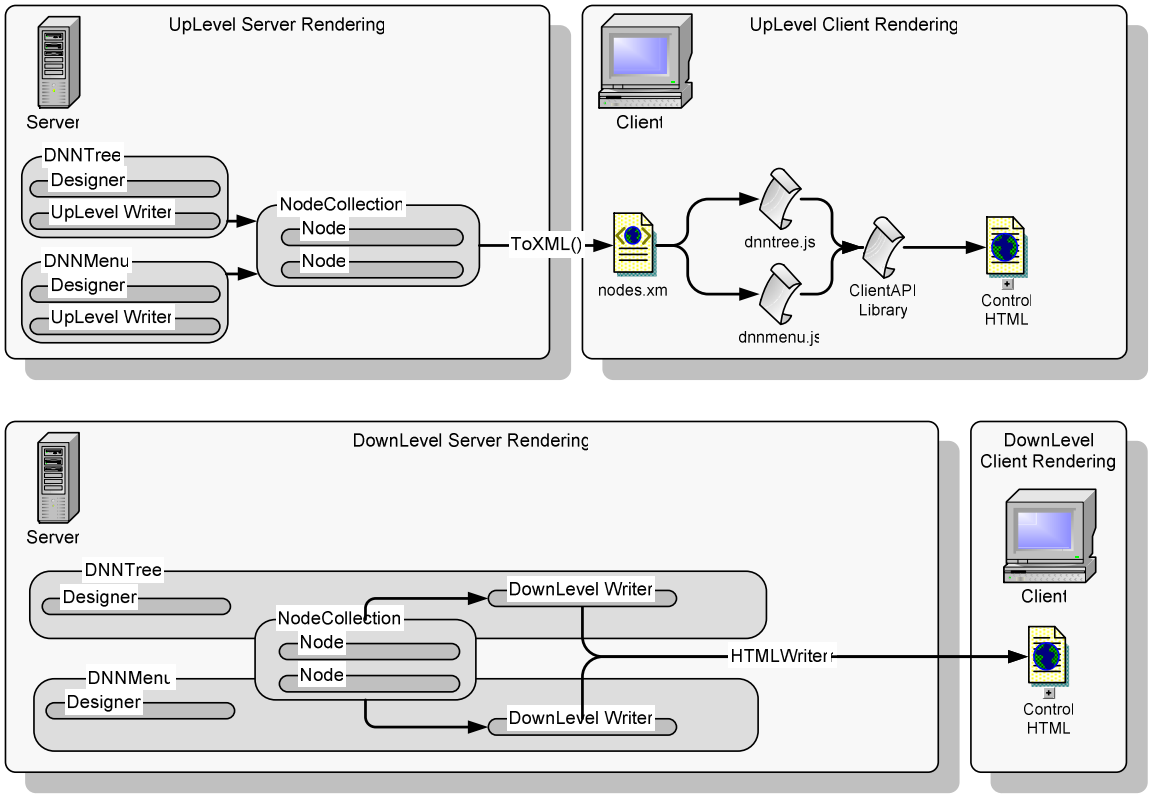


Article V. Consuming the DNNNodeCollection

As mentioned above, specialized nodes are intended to be created for the DotNetNuke WebControls by inheriting from the DNNNode object. This allows the DotNetNuke controls to easily consume the XML provided by the Navigation class. In addition, the

maintenance of the code is simplified since all the base functionality is inherited with each new webcontrol.

Having the hierarchy easily be represented with XML is also an advantage since the DotNetNuke controls are rendered by sending XML to the client. The following diagrams show both the UpLevel and DownLevel rendering of the controls.



Article VI. Navigation Class

Prior to 3.2, the navigation controls within DotNetNuke were populated by similar logic in each of the controls. This logic included tab attribute validation with things like permission, expiration dates, and visibility checks. It also included the filtering down of which nodes to display through the Level property (parent, child, root, and same). In version 3.2 we decided to increase the complexity of which nodes to display by introducing Populate On Demand.

Tab Attribute Validation Filtering

A simple helper function was made to do all the necessary validation checks.

```
Function IsTabShown(ByVal objTab As TabInfo, ByVal blnAdminMode As Boolean) As Boolean
```

First the tab's IsVisible and IsDeleted property are checked. Then one of two things needs to be true. The server's current date needs to fall between the tab's StartDate and EndDate or we need to be in AdminMode. Finally, the user must be authorized to see the tab (AuthorizedRoles).

<u>Level Filtering</u>

The primary function to retrieve the node collection has the following definition.

```
Public Shared Function GetNavigationNodes(ByVal objRootNode As DNNNode, ByVal eToolTips
As ToolTipSource, ByVal intStartTabId As Integer, ByVal intDepth As Integer, ByVal
intNavNodeOptions As Integer) As DNNNodeCollection
```

For Level filtering we primarily care about 2 parameters: `intStartTabId` and `intNavNodeOptions`.

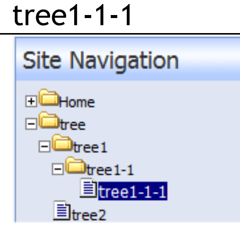
`intStartTabId` is to be at the root level of the hierarchy. If all nodes are to be returned a -1 is passed.

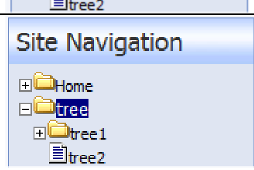
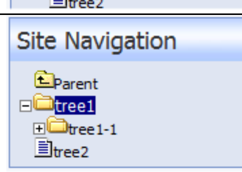
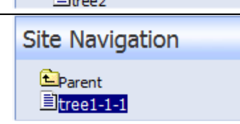
`intNavNodeOptions` is an integer that represents a bit value. An enumerator NavNodeOptions holds the different bit values.

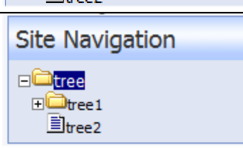
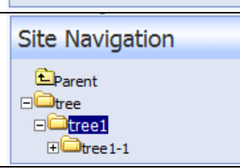
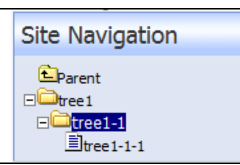
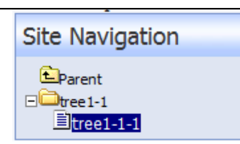
```
IncludeSelf = CInt(2 ^ 0)
IncludeParent = CInt(2 ^ 1)
IncludeSiblings = CInt(2 ^ 2)
```

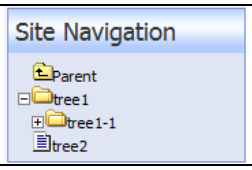
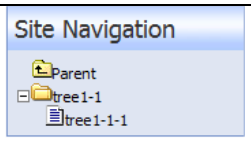
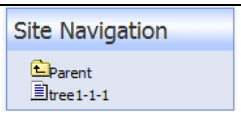
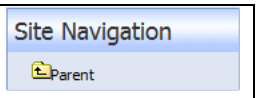
The following table displays a simple hierarchy that is shown in its entirety for the Root row. As each node is selected you can compare what the tree will look like for each level. Underneath each level name you will see the bit values that are passed into the `GetNavigationNodes` function.

Level	Tree	tree1	tree1-1	tree1-1-1

Root
`IncludeSelf +`
`IncludeSiblings`
				

Same
`IncludeSelf +`
`IncludeSiblings`
				

Parent
`IncludeSelf +`
`IncludeParent`
				

Child				

If it not already evident the NavNodeOptions passed in determine which nodes are displayed. `IncludeSelf` determines if the passed in `intStartTabId` will be displayed. This is the case in all Levels except for Child. The `IncludeSiblings` determines if the nodes that share the same parent as the `intStartTabId` node will displayed. Finally the `IncludeParent` bitvalue determines if the parent of the `intStartTabId` node should be displayed.

From the looks of the chart you may be asking what the difference is between the Root and Same Levels. They both pass in the same NavNodeOption of `IncludeSelf` and `IncludeSiblings`. The difference is the Root level always passes in a `intStartTabId` = -1 whereas the Same passes in the current Active TabId.

New: The skin object now supports the passing in of an arbitrary TabId to act as the root (`StartTabId`). This has been a common feature request of both the skin navigation objects and Solution Partners Menu Module.

Populate On Demand Filtering

For Populate On Demand (POD) we primarily care about a single parameter: `intDepth`. This parameter determines how many levels at a time each POD request will fetch. If a -1 is passed (as is the case of you have your usercontrol's `PopulateNodesFromClient` set to False) the entire node collection will be sent down. Inside the `GetNavigationNodes` method prior to adding a child node the following function is invoked.

```
Function IsTabPending(ByVal objTab As TabInfo, ByVal objParentNode As DNNNode, ByVal
objRootNode As DNNNode, ByVal intDepth As Integer, ByVal objBreadCrumbs As Hashtable,
ByVal intLastBreadCrumbId As Integer) As Boolean
```

This function is responsible for determining if the node should be added to the collection of if its parent should be marked as having a pending node (`HasNodes = True`). The equation for determining this is pretty straightforward.

1. If CurrentNodeLevel - RootNodeLevel <= intDepth Then the node should be added.
2. If Node is already part of the breadcrumbs then the node should be added.
3. Finally if The Parent node is part of the breadcrumb and it is not the last breadcrumb then add the node.
4. Otherwise, mark the node as pending.

If the 3. is not clear to you then please have a look at the Root Level grid above and say the tree1-1 node is selected. We need to show the tree2 node but not the nodes

under tree1-1, since they will be loaded with POD. This assumes your depth is 1 of course.

Article VII. Populate On Demand

One of the features added to the navigational DotNetNuke WebControls is the ability to load-on-demand. Similar to the tree included in ASP.NET 2.0, the DotNetNuke WebControls will utilize a client-side callback to retrieve the children. Since DNN will be compatible with ASP.NET 1.x for some time now, we decided to implement client-side capability into the ClientAPI in a manner very similar to what is included in ASP.NET 2.0. If you are interested with how the ClientAPI implements callbacks see the DotNetNuke_ClientAPI-ClientCallback.doc file for more information.

As with any advanced technique implemented, you need to consider your different use cases and try an make it as simple as possible for the end user (in this case the developer) to utilize your technique. This is not always a simple task, considering the complexities of writing cross-browser code. As always the core team has its eyes on the techniques put out by Microsoft and what we came up with is quite similar to how they handle populate on demand in their TreeView control in ASP.NET 2.0.

Lets take a look at the ASP.NET 2.0 TreeView's two use cases: DownLevel and UpLevel Populate-On-Demand.

DownLevel Populate-On-Demand

The tree has two properties we are interested in, `PopulateOnDemand` and `OnTreeNodePopulate`. The first is a boolean flag set on each node that needs the populate on demand functionality. The second is the event handler for the `TreeNodePopulate` event. When a node is expanded that has its `PopulateOnDemand` property set to true a postback is issued, which in turn calls the `OnTreeNodePopulate` event handler.

The method signature for the event is as follows.

```
Sub TreeView1_TreeNodePopulate(ByVal sender As Object, ByVal e As TreeNodeEventArgs)
```

The `TreeNodeEventArgs` has a property called `Node` that contains the node that caused the event to fire. The populating of the child nodes is as simple as calling the `Add` method on the node. The ensuing page refresh causes the newly added nodes to be rendered. Lets compare that to how the UpLevel callback is handled.

UpLevel Populate-On-Demand

Internal to the Tree control is the implementation of the `ICallbackEventHandler`. This code will receive the callback and eventually raise the `OnTreeNodePopulate` event, passing the Node object. The code to populate the child nodes will be exactly

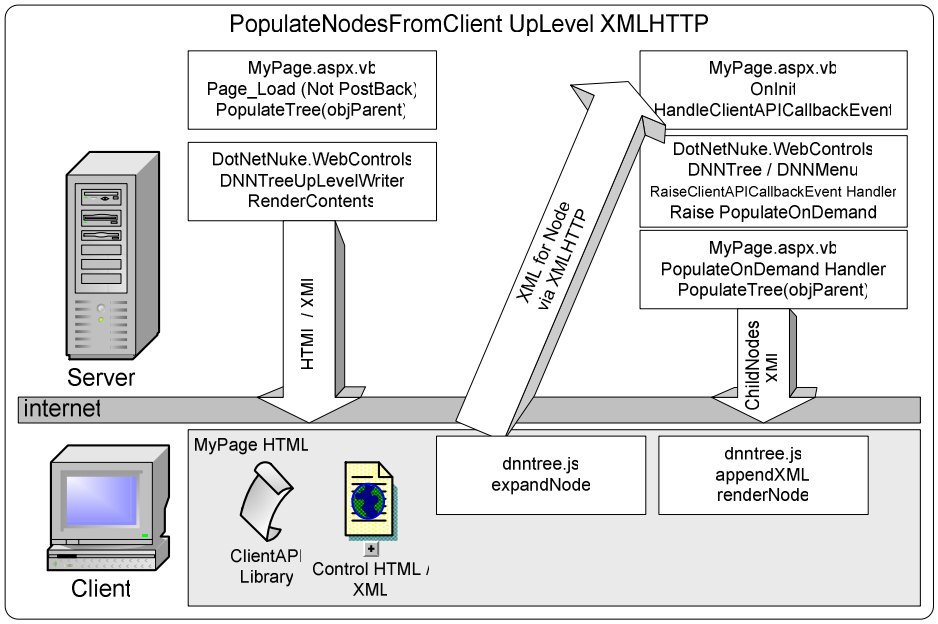
the same as the downlevel non-callback implementation. When the child nodes are populated the rest of the `ICallbackEventHandler` is run which generates the HTML for the newly added nodes and returns it to the client.

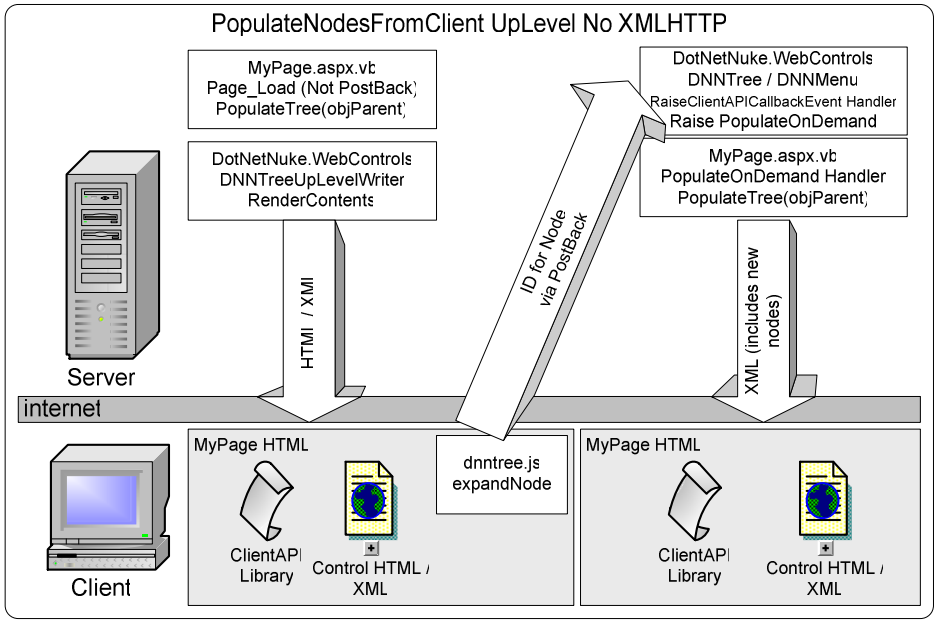
Almost the same...
When I originally coded the callback plumbing, I had the client only passing the id of the node that made the request. I had intended to have the code simply look up the node object in the DNNTree's node collection in order to pass it on the `TreeNodeEventArgs` parameter. This of course failed, since the node collection was not set. The reason for the failure is we are too early in the page's lifecycle for the node collection to be populated. In fact, we will never get to that stage in a callback, since it will "hijack" the request and terminate it once the callback completes. So the only way to successfully pass the node object in the `TreeNodeEventArgs` would be to send node's information serialized from the client. Luckily, in the DNNTree's implementation of this ends up being quite simple, since it has the xml representation available to it on the client. The ASP.NET 2.0 design is not so elegant. Basically they have to send a delimited string and parse it on the server.

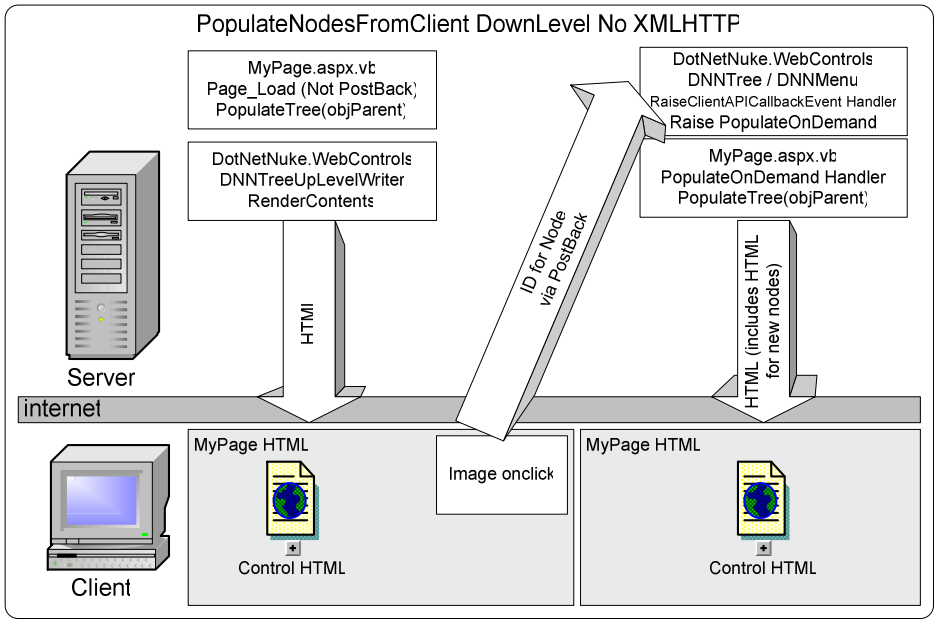
While this solution works, the end developer needs to understand that the only node available in the callback event will be the one making the request. In addition, the developer of the `TreeView1_TreeNodePopulate` event handler also cannot rely upon any programmatic settings that were applied to the tree. Failure to understand these differences will result in code that works in all the downlevel browsers and none of the uplevel ones. The good news is that the developer usually uses an uplevel browser to test, therefore will notice this right away.
I have considered making my implementation of the callback on the tree pass the entire xml node collection to the server, thus having the node collection populated, but decided not to for two reasons. The first is a performance consideration. Why send this potentially large string across the wire when it is most likely not needed. The second is I wanted my code to remain similar to what will be included in ASP.NET 2.0.

Unlike the ASP.NET Tree control which has 2 use cases, the DotNetNuke WebControls has at least three. This is due to the fact that the ClientAPI allows for its various namespaces to be included or excluded based on browser. So it is possible that the control will be rendered uplevel and not support XMLHTTP. **Note:** The ClientAPI XMLHTTP namespace actually supports **more browsers** than the Microsoft implementation does due to its ability to do callbacks via its javascript xmlhttp wrapper.

The following diagrams illustrate the three use cases that the DotNetNuke navigation webcontrols support.







It should be noted that a major difference between the DNNTree and the ASP.NET 2.0 implementation is what is being passed across the wire. Like I mentioned earlier, the initial request of the DNNTree is XML, whereas in the ASP.NET 2.0 implementation it is a custom delimited string. I believe XML here is the better choice, since it allows the node object to be easily eXtensible. Meaning if a property is added to the object, the client-side callback code does not need to be modified to include the new property in the delimited string. Additionally, the response of the callback in the ASP.NET 2.0 implementation is the HTML that is to be appended to the node making the request. In the DNNTree, it is the node's xml for its children. I believe that this payload will be much smaller than the HTML markup.

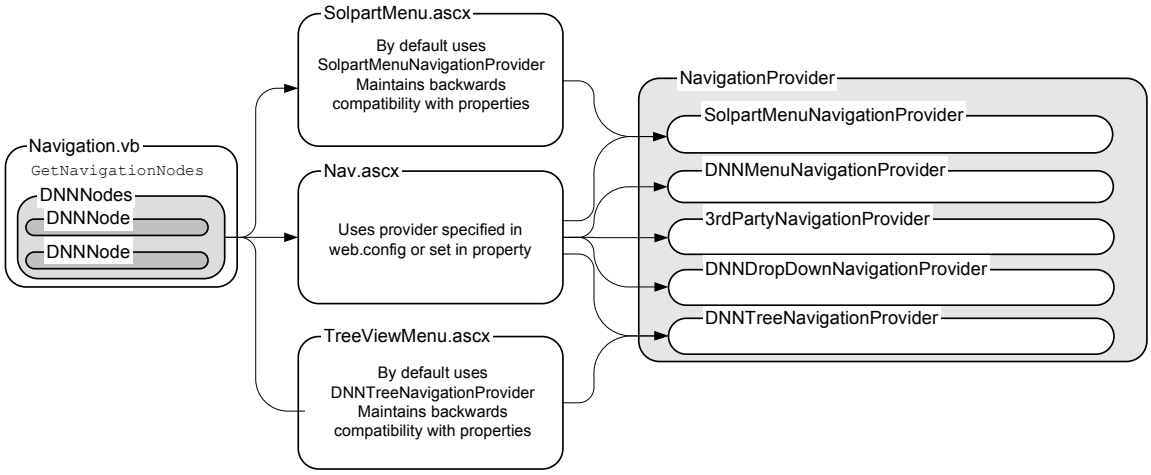
For more information on the ASP.NET 2.0 Tree control as it relates to callbacks see this article.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/treeview.asp

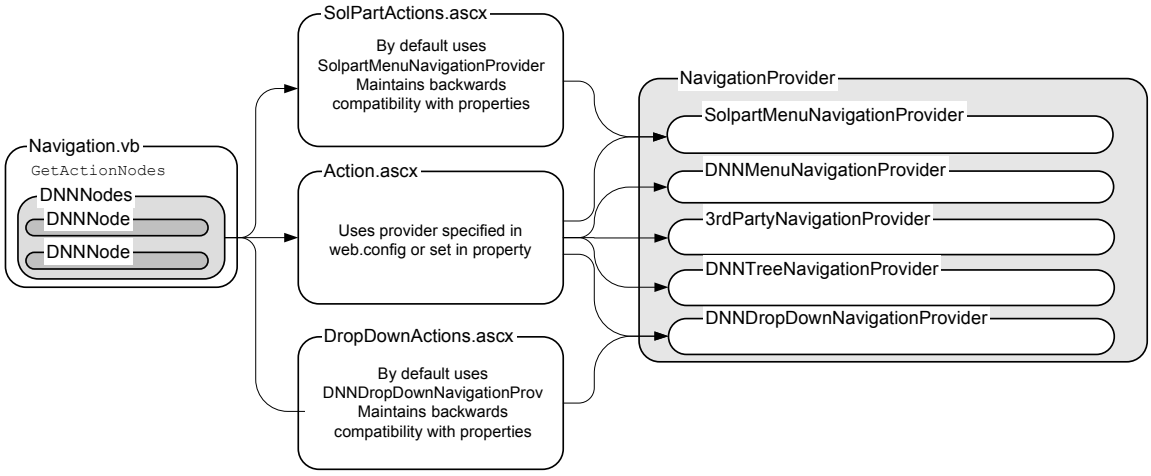
Article VIII.

Article IX. Navigation Provider / DotNetNuke Navigation UserControl Design

Since DotNetNuke 2.0 the menu that was utilized in DNN was "pluggable" by creating a new skin object. Each skin object was responsible to filtering out the nodes that should not be displayed. As mentioned earlier in this doc, this type of logic has been centralized into the Navigation class. To further the abstraction, it has been decided that the navigation components within DNN move towards a Provider pattern. The

following diagrams illustrate this abstraction. Notice that the central skin object in each diagram is generic and should be capable of handling any provider. Theoretically, all the skin objects could support any provider, but their underlying properties are named to match the control in which they implement. The generic skin objects will match the provider's interface names instead (See chart below).





Note: Action.ascx is not developed as of yet.

Another thing to note is that unlike most providers found in DNN, this model is intended to support multiple providers running at the same time. This will be determined by the skin object used through the new ProviderName property.

```
ProviderName="SolpartMenuNavigationProvider"
```

In order for a provider to function, a set interface must be decided upon. The primary design of the interface had two goals.

1. Preserve backwards compatibility
2. Name the properties in such a way that their intended purpose is more explicit.

The menu skin object that is defaulted within DNN was designed around the specific design of the Solution Partner's menu. While most of the properties exposed could apply to any hierarchical menu, there are some that more than likely will not. Ideally, these properties would be removed from the interface, but cannot without violating goal #1. So our interface will have a few more properties that would have been originally designed. The following chart lists out the interface along with its mapping to each of the providers.

DotNetNuke Navigation WebControls

	NavigationProvider	Description	SolpartMenuNavigationProvider	DNNMenuNavigationProvider	DNNTreeNavigationProvider	DNNDropDownNavigationProvider
Paths						
	PathImage	Directory to find the icons	✓			
	PathSystemImage	Directory to find the images necessary for the control (i.e. spacer.gif)	✓	✓	✓	
	PathSystemScript	Directory to find associated script files for the control	✓	✓	✓	
Rendering						
	ForceDownLevel	Flag to force the downlevel menu to display (values: true, false)	✓		✓	
	ForceCrawlerDisplay	Displays the full menu as an indented list of normal hyperlinks (like a sitemap) {true|false}	✓			
	ControlOrientation	Determines how the menu is displayed, horizontal or vertical (values: vertical, horizontal)	✓	✓		
	ControlAlignment	Alignment of the menu within the menu bar. {Left|Center|Right|Justify}	✓			
Mouse Properties						
	MouseOutHideDelay	Number of milliseconds to wait until menu is hidden on mouse out. (0 = disable)	✓			
	MouseOverDisplay	Adjusts effect when mouse moves over menu bar item (Values: Outset, Highlight, None)	✓			
	MouseOverAction	Makes menu expand on mouse over (unlike any menu found within the Windows environment) (Values: true, false)	✓			
Arrows						
	IndicateChildren	use arrows to indicate child submenus	✓	✓	✓	
	IndicateChildImageSub	Arrow image to use for sub menus - i.e. arrow.gif	✓	✓	✓	
	IndicateChildImage	Arrow image to use for root menus - i.e. arrow.gif	✓	✓	✓	
	IndicateChildImageExpandedSub	Image to use on sub node when children expanded			✓	
	IndicateChildImageExpandedRoot	Image to use on root node when children expanded			✓	
Custom HTML						
	NodeLeftHTMLRoot	HTML text added to the beginning of the Root menu items	✓			
	NodeRightHTMLRoot	HTML text added to the end of the Root menu items	✓			
	NodeLeftHTMLSub	HTML text added to the beginning of the sub menu items	✓			
	NodeRightHTMLSub	HTML text added to the end of the sub menu items	✓			
	NodeLeftHTMLBreadCrumbSub	HTML to display to the left of the node that resides in a submenu and is a breadcrumb	✓			
	NodeLeftHTMLBreadCrumbRoot	HTML to display to the left of the node that resides in the root and is a breadcrumb	✓			
	NodeRightHTMLBreadCrumbSub	HTML to display to the right of the node that resides in a submenu and is a breadcrumb	✓			
	NodeRightHTMLBreadCrumbRoot	HTML to display to the right of the node that resides in the root and is a breadcrumb	✓			
	SeparatorHTML	The separator between Root level menu items. This can include custom skin images, text, and HTML (ie. <![CDATA[<img src="file.gif">]]>)	✓	✓		
	SeparatorLeftHTML	The separator used just before a root level menu item. A use for this might be a left edge of a tab image for example	✓	✓		
	SeparatorRightHTML	The separator used just after a root level menu item. A use for this might be a right edge of a tab image for example	✓	✓		
	SeparatorLeftHTMLActive	The separator used just before an active root level menu item	✓	✓		
	SeparatorRightHTMLActive	The separator used just before an active root level menu item	✓	✓		
	SeparatorLeftHTMLBreadCrumb	The separator used just before a root level menu item found in the breadcrumb array list	✓	✓		
	SeparatorRightHTMLBreadCrumb	The separator used just before a root level menu item found in the breadcrumb array list	✓	✓		

CSS						
	CSSControl	Menu Bar CSS Class	✓	✓	✓	✓
	CSSContainerRoot	Menu Container CSS Class	✓			
	CSSContainerSub	SubMenu CSS Class	✓	✓		
	CSSBreak	Menu Break CSS Class	✓			
	CSSIndicateChildSub	Menu Arrow CSS Class	✓			
	CSSIndicateChildRoot	Menu Root Arrow CSS Class	✓			
	CSSNode	Menu Item CSS Class - ALL Nodes	✓	✓	✓	
	CSSNodeRoot	CSS Class used for root menu items - Overrides CSSNode	✓	✓	✓	
	CSSBreadCrumbSub	CSS Class used for sub menu items when they are found in the breadcrumb array list - Overrides CSSNode	✓	✓	✓	
	CSSBreadCrumbRoot	CSS Class used for root menu items when they are found in the breadcrumb array list - Overrides CSSNode	✓	✓	✓	
	CSSNodeSelectedRoot	CSS Class used for root menu items when they are the active tab - Overrides CSSNode, CSSBreadCrumbRoot	✓	✓	✓	
	CSSNodeSelectedSub	CSS Class used for sub menu items when they are the active tab - Overrides CSSNode, CSSBreadCrumbRoot	✓	✓	✓	
	CSSNodeHover	Menu Item CSS Class for mouse-over - ALL Nodes	✓	✓	✓	
	CSSNodeHoverSub	CSS Class used for sub menu items when they moused-over - Overrides CSSNodeHover	✓	✓	✓	
	CSSNodeHoverRoot	CSS Class used for root menu items when they moused-over - Overrides CSSNodeHover	✓	✓	✓	
	CSSSeparator	CSS class used for the root level menu item separator	✓	✓		
	CSSLeftSeparator	CSS class used for leftseparator	✓	✓		
	CSSRightSeparator	CSS class used for leftseparator	✓	✓		
	CSSLeftSeparatorSelection	CSS class used for leftseparatoractive	✓	✓		
	CSSRightSeparatorSelection	CSS class used for rightseparatoractive	✓	✓		
	CSSLeftSeparatorBreadCrumb	CSS class used for leftseparatorbreadcrumb	✓	✓		
	CSSRightSeparatorBreadCrumb	CSS class used for rightseparatorbreadcrumb	✓	✓		
	CSSIcon	Menu Icon CSS Class	✓	✓	✓	
Styles						
	StyleBackColor	Background color	✓			
	StyleForeColor	Fore color of menu item when selected	✓			
	StyleHighlightColor	Color of top and left border to give a highlight effect	✓			
	StyleIconBackColor	Background color in area where icon is displayed	✓			
	StyleSelectionBorderColor	Color of border surrounding selected menu item	✓			
	StyleSelectionColor	Background color of menu item when selected	✓			
	StyleSelectionForeColor	Fore color of menu item when selected	✓			
	StyleControlHeight	Menu bar height in pixels	✓			
	StyleBorderWidth	Menu border width in pixels	✓			
	StyleNodeHeight	Menu item height in pixels	✓			
	StyleIconWidth	Width of icon column in pixels	✓			
	StyleFontNames	Fonts to use	✓			
	StyleFontSize	Font Size Style	✓			
	StyleFontBold	Font Bold Style	✓			
Animation						
	EffectsShadowColor	Color of the shadow	✓			
	EffectsTransition	Determines which transition to display	✓			
	EffectsDuration	Number of seconds the transition will take	✓			
	EffectsShadowDirection	Determines which direction the shadow will fall	✓			
	EffectsShadowStrength	Determines how opaque the shadow is	✓			
	EffectsStyle	IE only property for SubMenu styles and transitions	✓			

You will notice that most properties have been grouped into the following categories (via a prefix).

- CSS – CSS classname to assign
- Style – Style attribute to assign
- Path – Path/Directory to locate images and script
- Mouse – Settings to handle Mouse specific events
- Node – Properties like custom HTML to apply to specific nodes
- Separator – Properties like custom HTML to apply to separators
- Effects – Effects to apply to the menu

The following chart attempts to show how the provider properties map to the original SolpartMenu skin object.

* Denotes Optional

	NavigationProvider	Description	SolpartMenu.ascx Property	SolpartMenu.ascx Notes
Control				
	NavigationControl	hierarchy		
	ControlID	ID of control		
Paths				
	PathImage	Directory to find the icons		PortalSettings.HomeDirectory
		skin object will toggle ImageDirectory based on value	useskinpatharrowimages	Use arrow images located in the skin and not those in the /images folder {true|false}
	PatySystemImage	Directory to find the images necessary for the control (i.e. spacer.gif)		PortalSettings.ActiveTab.SkinPath else Common.Globals.ApplicationPath & "/images/"
	PathSystemScript	Directory to find associated script files for the control		"/controls/SolpartMenu/"
Control Rendering				
	ForceDownLevel	true, false)	forcedownlevel	
	ForceCrawlerDisplay	Displays the full menu as an indented list of normal hyperlinks (like a sitemap) {true|false}	forcefullmenulist	
	ControlOrientation	Determines how the menu is displayed, horizontal or vertical (values: vertical, horizontal)	display	
	ControlAlignment	{Left|Center|Right|Justify}	menualignment	
Mouse Properties				
	MouseOutHideDelay	Number of milliseconds to wait until menu is hidden on mouse out. (0 = disable)	menueffectsmouseouthidedelay, mouseouthidedelay	
	MouseOverDisplay	Adjusts effect when mouse moves over menu bar item (Values: Outset, Highlight, None)	menueffectsmouseoverdisplay	
	MouseOverAction	menu found within the Windows environment) (Values: true, false)	menueffectsmouseoverexpand	
Arrows				
	IndicateChildren	use arrows to indicate child submenus	usearrows	
		skin object will assign IndicateChildImage/IndicateChildImageRoot	downarrow	arrow image used for downward facing arrows indicating child submenus. if not specified defaults to menu_down.gif
		skin object will assign IndicateChildImage/IndicateChildImageRoot	rightarrow	arrow image used for right facing arrows indicating child submenus (defaults to breadcrumb.gif)
	IndicateChildImageSub	Arrow image to use for sub menus - i.e. arrow.gif		If userarrows then assign value in rightarrow
	IndicateChildImage	Arrow image to use for root menus - i.e. arrow.gif		if userarrows and vertical orientation use rightarrow if usearrows and horizontal orientation use downarrow
	IndicateChildImageExpandedSub	Image to use on sub node when children expanded		
	IndicateChildImageExpandedRoot	Image to use on root node when children expanded		
Custom HTML				
	NodeLeftHTMLRoot	items	rootmenuitemlefthtml	
	NodeRightHTMLRoot	HTML text added to the end of the Root menu items	rootmenuitemrighthtml	
	NodeLeftHTMLSub	items	submenuitemlefthtml	
	NodeRightHTMLSub	HTML text added to the end of the sub menu items	submenuitemrighthtml	
		skin object will assign NodeLeftHTMLBreadCrumb	usesubmenubreadcrumbarrow	tabs that are listed in the breadcrumb array list {true|false}
		skin object will assign NodeLeftHTMLBreadCrumbRoot	userootbreadcrumbarrow	Use a breadcrumb arrow to identify the root tab that is listed in the breadcrumb array list {true|false}
	NodeLeftHTMLBreadCrumbSub	HTML to display to the left of the node that resides in a submenu and is a breadcrumb		usesubmenubreadcrumbarrow flag, assigns IMG SRC=ArrowImageBreadCrumb
	NodeLeftHTMLBreadCrumbRoot	HTML to display to the left of the node that resides in the root and is a breadcrumb		userootbreadcrumbarrow flag, assigns IMG SRC=ArrowImageBreadCrumbRoot
	NodeRightHTMLBreadCrumbSub	HTML to display to the right of the node that resides in a submenu and is a breadcrumb		
	NodeRightHTMLBreadCrumbRoot	HTML to display to the right of the node that resides in the root and is a breadcrumb		
	SeparatorHTML	can include custom skin images, text, and HTML (ie. <![CDATA[<img src="file.gif">]]>)	separator	if finds src=, it prefixes with PortalSettings.ActiveTab.SkinPath
	SeparatorLeftHTML	item. A use for this might be a left edge of a tab image for example	leftseparator	if finds src=, it prefixes with PortalSettings.ActiveTab.SkinPath
	SeparatorRightHTML	A use for this might be a right edge of a tab image for example	rightseparator	if finds src=, it prefixes with PortalSettings.ActiveTab.SkinPath
	SeparatorLeftHTMLActive	menu item	leftseparatoractive	PortalSettings.ActiveTab.SkinPath
	SeparatorRightHTMLActive	menu item	rightseparatoractive	PortalSettings.ActiveTab.SkinPath
	SeparatorLeftHTMLBreadCrumb	The separator used just before a root level menu item found in the breadcrumb array list	leftseparatorbreadcrumb	if finds src=, it prefixes with PortalSettings.ActiveTab.SkinPath
	SeparatorRightHTMLBreadCrumb	The separator used just before a root level menu item found in the breadcrumb array list	rightseparatorbreadcrumb	if finds src=, it prefixes with PortalSettings.ActiveTab.SkinPath

CSS				
	CSSControl	Menu Bar CSS Class	menubarcssclass	defaults to MainMenu_MenuBar if not specified
	CSSContainerRoot	Menu Container CSS Class	menucontainercssclass	defaults to MainMenu_MenuContainer if not specified
	CSSContainerSub	SubMenu CSS Class	submenucssclass	defaults to MainMenu_SubMenu if not specified
	CSSBreak	Menu Break CSS Class	menubreakcssclass	defaults to MainMenu_MenuBreak if not specified
	CSSIndicateChildSub	Menu Arrow CSS Class	menuarrowcssclass	defaults to MainMenu_MenuArrow if not specified
	CSSIndicateChildRoot	Menu Root Arrow CSS Class	menurootarrowcssclass	specified
	CSSNode	Menu Item CSS Class - ALL Nodes	menuitemcssclass	defaults to MainMenu_MenuItem if not specified
	CSSNodeRoot	CSSNode	rootmenuitemcssclass	
	CSSBreadCrumbSub	found in the breadcrumb array list - Overrides CSSNode	submenuitembreadcrumbcssclass	
	CSSBreadCrumbRoot	found in the breadcrumb array list - Overrides CSSNode	rootmenuitembreadcrumbcssclass	
	CSSNodeSelectedRoot	the active tab - Overrides CSSNode, CSSBreadCrumbRoot	rootmenuitemactivecssclass	
	CSSNodeSelectedSub	the active tab - Overrides CSSNode, CSSBreadCrumbRoot	submenuitemactivecssclass	
	CSSNodeHover	Menu Item CSS Class for mouse-over - ALL Nodes	menuitemselcssclass	defaults to MainMenu_MenuItemSel if not specified
	CSSNodeHoverSub	CSS Class used for sub menu items when they moused-over - Overrides CSSNodeHover	submenuitemselectedcssclass	
	CSSNodeHoverRoot	CSS Class used for root menu items when they moused-over - Overrides CSSNodeHover	rootmenuitemselectedcssclass	
	CSSSeparator	separator	separatorcssclass	
	CSSLeftSeparator	CSS class used for leftseparator	leftseparatorcssclass	
	CSSRightSeparator	CSS class used for leftseparator	rightseparatorcssclass	
	CSSLeftSeparatorSelection	CSS class used for leftseparatoractive	leftseparatoractivecssclass	
	CSSRightSeparatorSelection	CSS class used for rightseparatoractive	rightseparatoractivecssclass	
	CSSLeftSeparatorBreadCrumb	CSS class used for leftseparatorbreadcrumb	leftseparatorbreadcrumbcssclass	
	CSSRightSeparatorBreadCrumb	CSS class used for rightseparatorbreadcrumb	rightseparatorbreadcrumbcssclass	
	CSSIcon	Menu Icon CSS Class	menuiconcssclass	defaults to MainMenu_MenuItem if not specified
Styles				
	StyleBackColor	Background color	backcolor	
	StyleForeColor	Fore color of menu item when selected	forecolor	
	StyleHighlightColor	Color of top and left border to give a highlight effect	highlightcolor	
	StyleIconBackColor	Background color in area where icon is displayed	iconbackgroundcolor	
	StyleSelectionBorderColor	Color of border surrounding selected menu item	selectedbordercolor	
	StyleSelectionColor	Background color of menu item when selected	selectedcolor	
	StyleSelectionForeColor	Fore color of menu item when selected	selectedforecolor	
	StyleControlHeight	Menu bar height in pixels	menubarheight	
	StyleBorderWidth	Menu border width in pixels	menuborderwidth	
	StyleNodeHeight	Menu item height in pixels	menuitemheight	
	StyleIconWidth	Width of icon column in pixels	iconwidth	
	StyleFontNames	Fonts to use	fontnames	
	StyleFontSize	Font Size Style	fontsize	
	StyleFontBold	Font Bold Style	fontbold	
Animation				
	EffectsShadowColor	Color of the shadow	menueffectsshadowcolor	
	EffectsTransition	Determines which transition to display	menueffectsmenutransition	AlphaFadeBottomRight, Barn, Blinds, Checkerboard, ConstantWave, Fade, GradientWipe, Inset, Iris, RadialWipe, Random, RandomBars, Slide, Spiral,
	EffectsDuration	Number of seconds the transition will take	menueffectsmenutransitionlength	
	EffectsShadowDirection	Determines which direction the shadow will fall	menueffectsshadowdirection	Supports: None, Top, Upper Right, Right, Lower Right, Bottom, Lower Left, Left, Upper Left
	EffectsShadowStrength	Determines how opaque the shadow is	menueffectsshadowstrength	
	StyleMenuEffects	IE only property for SubMenu styles and transitions	menueffectsstyle	
Misc				
		skin object decides if defaults should be populated	cleardefaults	settings of the menu so that they can be left empty and not just overridden with another value. Clears defaults for SelectedBorderColor, SelectedForeColor, HighlightColor, IconBackgroundColor, ShadowColor, SelectedColor, BackColor, ForeColor
			separatecss	Use CSS defined in a style sheet (values: true, false) Menu will not emit its own CSS styles
		Call to Navigation class passes in which nodes to populate	level	Root level of the menu in relationship to the current active tab {Root|Same|Child}
			rootonly	indicator to turn off submenus {true|false}
		Call to Navigation class passes in which property to populate the node's tooltip property	tooltip	Tooltips added to the menu items. These come from the tab object properties which are filled from the tabs table {Name|Title|Description}
			moveable	obsolete

NavObjectBase Class

One thing that may not be apparent to the average developer, for it was not apparent to me at first, is the need to have a "placeholder" for the skin's properties until the underlying control is created. We cannot have a skin object pass in a property like ProviderName to use and in the same set of properties assign the values directly to that object (i.e. CSSNode). The object is not created until the initialization event of the UserControl fires, whereas, the assignment of the properties happen before this. This is one of the many reasons for creating an underlying base class (NavObjectBase.vb) for which the SolpartMenu and TreeViewMenu skin objects inherit. This base class contains all the properties that the navigation provider supports. It is responsible for the assignment of the page properties to the provider's properties (when instantiated).

In addition to working around the issue above, this base class also provides another advantage. It allows the "old" skin objects to support both their preexisting properties and the new properties with little effort. This should allow for an easier transition towards the new interface, and eventually the new Nav skin object.

The last thing the base class is used for is a common place to put properties that are not needed by the provider's interface, but instead needed to call the GetNavigationNodes method. This includes properties for Populate On Demand (PopulateNodesFromClient, ExpandDepth), which nodes to display (Level, RootOnly, StartTabId), and what text to use for Tooltips (ToolTip). See the section titled Navigation Class for more information on each of these properties.

Nav.ascx UserControl

As previously mentioned, it is recommended that new skins created move away from the "old" skin objects and instead utilize the new Nav skin object. Its properties have been organized in a manner that should be easier to understand. Additionally, it should offer greater control over how the navigation control gets rendered. For example, the existing SolpartMenu skin object has a lot of value pre-defaulted. A simple declaration of the skin object like this

```
<dnn:MENU runat="server" id="dnnMENU"/>
```

will have an equivalent in the Nav control of this

```
<dnn:NAV runat="server" id="dnnNav" CSSControl="MainMenu_MenuBar"
CSSContainerRoot="MainMenu_MenuContainer" CSSNode="MainMenu_MenuItem"
CSSIcon="MainMenu_MenuIcon" CSSContainerSub="MainMenu_SubMenu"
CSSBreak="MainMenu_MenuBreak" CSSNodeHover="MainMenu_MenuItemSel"
CSSIndicateChildSub="MainMenu_MenuArrow" CSSIndicateChildRoot="MainMenu_RootMenuArrow"
PathSystemImage="[APPIMAGEPATH]" PathImage="[HOMEDIRECTORY]" MouseOverDisplay="highlight"
ProviderName="SolpartMenuNavigationProvider"
NodeLeftHTMLBreadCrumbRoot="<img src='[APPIMAGEPATH]/breadcrumb.gif'>" />
```

As you can tell the SolpartMenu skin object was defaulting a lot of CSS class names, paths, and HTML. Some properties like the UseBreadcrumbArrow and

RootBreadCrumbArrow are no longer available, and no longer defaulted (obviously). In its place is the ability to specify ***any*** html you wish to denote the root breadcrumb on either the left or the right side (NodeLeftHTMLBreadCrumbRoot/NodeRightHTMLBreadCrumbRoot).

Another thing to notice in the sample is the ability to have tokens defined in your properties to utilize some dynamic paths. The following chart details the currently supported tokens for all Path and HTML properties.

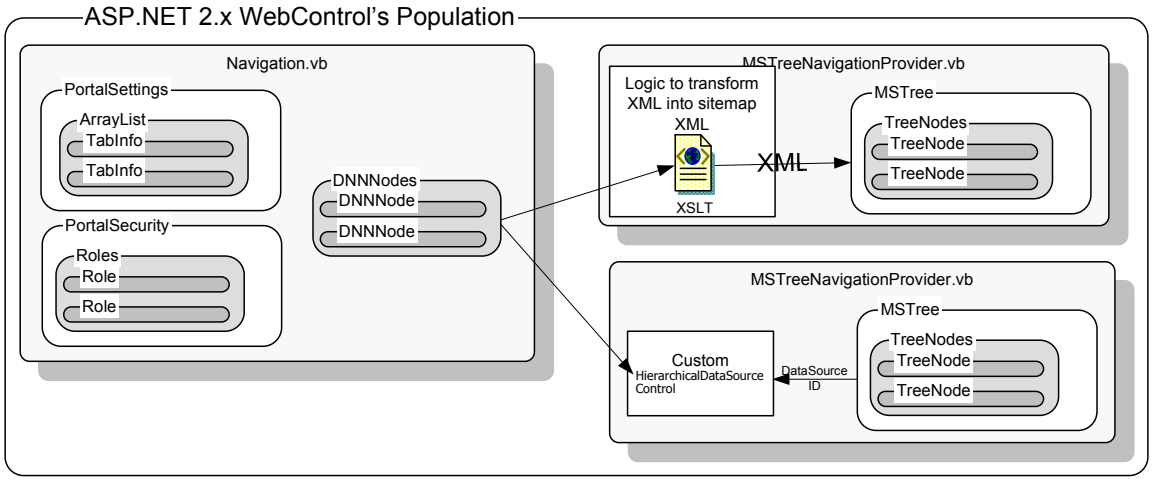
[SKINPATH]	PortalSettings.ActiveTab.SkinPath
[APPIMAGEPATH]	Common.Globals.ApplicationPath & "/images/"
[HOMEDIRECTORY]	PortalSettings.HomeDirectory
~	Any path prefixed with a ~ will be resolved through the ResolveURL method.

Other properties like RootOnly are also going to be depreciated. For to accomplish the same thing now you would use a combination of new properties.

StartTabId = -1 (Start tab is the root)
ExpandDepth = 1 (We want a single level to be pulled)
PopulateNodesFromClient = False (If we did not have this on the root nodes with children would be fetched "on demand")

Article X. ASP.NET Navigation UserControl Design

The upcoming version of ASP.NET will incoperate a concept called DataSources for its navigation controls. The SiteMapDataSource and XMLDataSource are the primary objects used for binding. Both of these objects inherit from the HierarchicalDataSourceControl object. By overriding some simple methods we can create our own. This gives us two options to allow the DNNNodeCollection to be digested by the ASP.NET navigation controls: XMLDataSource or CustomDataSource.



XMLDataSource

These DataSources provide the ASP.NET navigation controls a hierarchy of node objects looking something like this.

```
<siteMapNode title="Book 0" url="~/contentPages/book0/book.aspx" description="Go To Book 0"/>
```

The DNNNodeCollection, being an XML Document, contains its own way to represent its node hierarchy.

```
<n url="http://localhost/DotNetNuke/Home/tabid/36/Default.aspx" id="36" text="Home" key="36"
clickAction="3" />
```

The fact that they represent their underlying Node object differently is not something to worry about, for controls like the XMLDataSource have a property that allows an XSL Transformation to take place. Thus with little effort the new ASP.NET controls can be utilized with the DNNNodeCollection.

CustomDataSource

The option to create a custom datasource is probably the most elegant solution. Simply inherit from the HierarchicalDataSourceControl object, override a few methods, add the control to the page, and reference the datasource in your hierarchical control through the DataSourceID property.

```
<%@ Register Namespace="DotNetNuke.UI" TagPrefix="DNN" %>
...

<DNN:DNNNodeDataSource ID="DNNNodeDataSource1" runat="server" />
<asp:TreeView ID="TreeView1" runat="server" DataSourceID="DNNNodeDataSource1" ExpandDepth="0">
  <DataBindings>
    <asp:TreeNodeBinding DataMember="SiteMapNode" NavigateUrlField="url" TextField="text" />
  </DataBindings>
</asp:TreeView>
```

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	Aug 16, 2005	Shaun Walker	• Applied new template