Snapsis CSS NavMenu Development Guide

Overview

This document outlines the Snapsis NavMenu Skin Object for the DotNetNuke® portal system. This module supports any type of navigation through a template driven configuration, including built-in Tab Style navigation, Hover - Flyouts, Vertical or Horizontal orientations, and a unique select-drop down list box option.

- Strong focus on using CSS for graphics reference and style will reduce your development time and enable your end users to have more flexibility in getting the exact look they want without modifying your layout files. Most if not all changes to the look can be made in the CSS. If you use a template you can also configure layout & markup of the output.

- Several Sample Skins are included and are easy to copy and modify to make your own skins with this control.
    a. You can see the skin samples online at http://demo.snapsis.com

This navigation system was designed to be very flexible and extensible. It also promotes good SEO so that your sites can get noticed by the major search engines. The guidelines laid out in this document are meant to assist, but creativity should be a goal, so please let us know if the system is limiting in any way, or if you have ideas for future enhancements by visiting http://www.snapsis.com/support.aspx.

How it works

The tabbed solution was modeled after the article by Daniel Bowman at http://www.alistapart.com/articles/slidingdoors/. It is a very good approach to using pure CSS for displaying navigation links as unordered lists. The basic idea is to create a background image that is very large and slice it so that when applied to the background of two different elements in the html and positioning the background with CSS. This large image is then able to "expand" and contract by showing more or less of the background depending on the size and amount of text in the link. To use the tabbed solution you set your NavType property in the menu to "Tabs".

The hover-flyout solution is modeled after the Son of Suckerfish using embedded Unordered Lists <UL>. This solution does not require any external JS. The mouse-overs for IE browsers are added inline. Set navType="Hover" for this type of menu.

The Template solution allows you to configure the complete output of the menu including layout, markup, styles, etc. This type of menu can be used to make virtually any type of menu system. Set NavType="Template" and the TemplatePath="YourTemplateFile" to use this menu.

Using the NavMenu in your own skins

To use the object in your own skins you only need to register it and then include the tag that calls it in your html:

```
<%@ Register TagPrefix="Snapsis" Namespace="Snapsis.DNN.Skins" Assembly =
                        "Snapsis.DNN.NavMenu" %>

        <Snapsis:NavMenu id="ParentTabs" Level="0-0" Type="Tabs"
                                ExcludeTabs="Admin,Host" runat="server" />
```

The above two lines will display the top level (parent) tabs from the portal. To include additional levels, you place additional tags in your html that identify the levels that you want displayed, thereby creating multiple instances of the menu control The next line will display the first through third level under the parent tabs (which were at level zero):

```
<Snapsis:NavMenu id="ChildTabsLevel1" Level="1-3" Type="Tabs"
ExcludeTabs="Admin,Host" runat="server" />
```

Always specify the **Level** with a starting and ending range, even if it is only one level that you want to display. Along with the Level property, there are additional properties to control the NavType of the menu, along with **ExcludeTabs** and **IncludeTabs** which will assist in keeping only the tabs that you want to be rendered on that specific instance.

Property Definitions

The definitions of the properties and their values are described below:

ID- Set this property to place an ID on the rendered control. This is needed to reference the different instances of the control in the CSS file.

Level – A number from 0 to n to identify what level of tabs should be displayed. Specify this setting with a starting and ending level e.g Level="0-0" will only display the root tabs. Level="1-3" will display the 1st through 3rd levels.

NavType –

> **Tabs-** Will display a list of your links that can be configured to look like tabs with the background-images that are defined in the CSS file. The vertical or horizontal orientations are accomplished through CSS mostly by setting a width on the main container.
>
> **ActiveChildTabs-** Used for displaying a list of tabs that has the currently active tab as a parent.
>
> **Hover –** Used if you want to set the menu to use embedded Unordered Lists with detection to add the mouse-overs for IE browsers.
>
> **SelectList-** Can be used to display all of the tabs in a single select box. It's useful for displaying the admin and host tabs since those tabs have long names that are not displayed well in a tab form.
>
> **Template-** The template type of menu allows you to specify an external template file so that you can completely configure the output of the menu including layout, styles, etc.

IncludeTabs- A list of tab names separated by commas that will be the only ones loaded in this instance of the control.

ExcludeTabs- A list of tab names separated by commas that will be excluded from all the other tabs that are loaded in this instance of the control.

> **Note: you can also use regular expressions to define includes and excludes for your menus.

CacheTabs – If set to true will cache the tab generation process so that the menu does not have to be rebuilt on every request. Set this to true after you have your menu working like you want.

DisplayIcon – If set to true will use the Icon file that is assigned to the Page in Page Settings.

HideTabName - If set to true will hide the name of the tab. Usually used with the DisplayIcon setting to make image type menus.

HideTabNamesByLevel - Set this to a range of levels that you want to hide tab names on. Useful for displaying Images on the parent, and still showing the names on child menus.

ShowHiden - If set to true will also include tabs/pages in the menu that have been set to hidden in the Page Settings of DotNetNuke.

OnlyExpandActive – used rarely, set this to true if you want the child items in a single in a single instance to only be shown after the parent is clicked on. Usually used in a vertical navigation to give the navigation the feel of an expanding type tree.

DisplayRoles- A list of security role names separated by commas that will be allowed to see this instance of the menu. This allows you to display different menus for different roles. This property can also use RegEx wildcards.

Example: DisplayRoles="IT_.*" will display for any user in a Role that begins with "**IT_"**).

DisplayPaths- A list of url paths separated by commas that will be allowed to see this instance of the menu. This allows you to display different menus for different urls. This property can also use RegEx

Example: "forums.*, will allow this instance of the menu to only be displayed if the url has "Forums" in it).

Replacements- A comma separated list of search=replace pairs. This is useful if you don't like something about the output of the menu, but don't have access to change it through some other property. This property can also use RegEx.

Example: If you wanted to replace the word Home with the word Start you would use Replacements="Home=Start".

TemplateFile- Specify the Filename of the Template to be used for this instance.

ShowType – Used very rarely. This property was added to support using two instances of the menus on the same page as in the case when you want to display root navigation in one instance and child navigation in another, with the child instance only displaying the links for the currently active root.
**Note: This setting is only valid when using the "Tabs" NavType.

WithActiveParent- This value will render all the active children of the currently active tab. If you are using two instances of the menu (one for parent and one for child, then you will most likely set ShowType="WithActiveParent" on your child menu.

ChildrenOnly – This value will make the menu show only the children of the tabs specified in the **IncludedTabs** property. It is useful if you want to target a specific section of the tab hierarchy.

Always- This will make the menu always render whatever levels are specified. It is the default and is not normally set explicitly.

LocalizeTabs – If you have a copy of the PageLocalization module from Apollo Software, you can set this property to true and the tabs will be localized.
http://www.snowcovered.com/Snowcovered2/Default.aspx?tabid=166&CatalogItemID=1616

Creating Hover Flyout navigation (suckerfish)

The Hover type of navigation is best explained by reading the article by Patrick Griffiths and Dan Webb
At the following link: Son of Suckerfish drop-downs

The Snapsis NavMenu creates this structure automatically from the DotNetNuke page hierarchy and also detects IE browsers to include the needed mouse-over effects so no external JavaScript is needed.

Example CSS used in Hover type menu samples:

```
#nav ul {
    padding:0;
    margin:0;
    list-style:none;
    float:left;
    width:auto;
    background-color:#6CC1F7;
}

#nav li {
  position: relative;
  float:left;
  line-height : 1.25em;
  width: 9em;
  list-style: none;
}

#nav li ul { /* second-level lists */
          position : absolute;
          left: -999em;
          width:13em;
    background-color:#6CC1F7;
          border:1px solid #809DF9;
}


#nav a {
    font-size:1.08em;
          display:block;
    text-decoration:none;
    text-align:center;
    font-weight:bold;
    width:8.5em;
    padding :2px 0 2px 0;
    color:#000;
    border:1px solid #809DF9;
    background-color:#6CC1F7;
}
#nav li li a {
```

```
    width:12em;
}

#nav li a:hover {
  background-color: #fff;
  border:1px solid #809DF9;
}
#nav li a.SelectedTab {
    font-size:1.12em;
    background-color: #FFF;
}
#nav .DisabledTab {
    color:#ccc;
}
/* no need to change these hover classes
unless you need more than 4 levels */
#nav li:hover ul ul,
#nav li:hover ul ul ul,
#nav li:hover ul ul ul ul,
#nav li.iehover ul ul,
#nav li.iehover ul ul ul,
#nav li.iehover ul ul ul ul{
  left: -999em;
}

#nav li:hover ul,
#nav li.iehover ul {
            top: 1.6em;
            left:0;
}
#nav li li:hover ul,
#nav li li li:hover ul,
#nav li li.iehover ul,
#nav li li li.iehover ul {
            top: 0;
            left:12.8em;
}
```

Creating Tab Navigation (sliding doors)

Using the default skin that comes with this module is a great way to understand how it all works.

The sample CSS Tabs skins are included as an individual zip files and can be uploaded through the skin uploader, or just unzipped directly and transferred to your portal's skin folder.

Creating your own Tab Graphics

The tab graphics are 300px X 300px and contain both the mouseover and the normal version all in one image. Slice the image on the left wide enough to get any curve that might be needed. The padding for the TabList Li class should be set to the same width as that of the sliced left image.
Study the images with the included skin to get an idea how to make them. There is also a photoshop PSD file included to help in building your own tab graphics.

Explanation of the CSS

The CSS selectors defined below are used for changing the presentation of the NavMenu object when using the Tabs menu type:

```
/*   The csshover.htc Credit: Peterned - http://www.xs4all.nl/~peterned/  */

                body{
                behavior:url("DesktopModules/SnapsisDNN/csshover.htc");
                height:100%;
                color:#000000;
                }
                /* The following two selectors attempt to
                make the content area fill the screen */

                html{height:100%}
                html>body #PortalContainer {height:auto}/*for mozilla */

                /* This selector is the main container for the List of links */
                .TabList {
                float:left;
                horizontal-align:center;
                }

                /* The next two selectors cause the list to
                be displayed in a horizontal line without bullets */
                .TabList ul {
                clear:both;
                margin:0;
                padding:10px 10px 0;
                list-style:none;
                display: inline;
                }

                .TabList li {
                float:left;
                margin:0;
```

```
display: inline;
padding:0 0 0 2px;
}

/* the font is set in the
anchor elements which are in each list element */

.TabList li a {
font-family:Tahoma,San-Serif;
font-size:10pt;
voice-family: "\"}\""; voice-family:inherit;
line-height:normal;
}

/* This selector sets the style for the SelectList Menu Type */
select.TabList {
border-left: #003366 1px solid;
border-right: #003366 1px solid;
border-top:  #003366 1px solid;
border-bottom: #003366 1px solid;
background-color: #003366;
color:#FFFFFF;
font-weight:bold;
font-size:80%;
}

/* IE-PC doesn't see these
child selectors but CSS2 compliant browsers can make use of them

#ParentTabs > ul a {width:auto;}
#ChildTabsLevel1 > ul a {width:auto;}

/* Commented Backslash Hack hides rule from IE5-Mac \*/
#ParentTabs a {float:none;}
#ChildTabsLevel1 a {float:none;}
/* End IE5-Mac hack */

/*The container tab for the
Parent Tabs is mostly used for positioning */

#ParentTabsContainer {
position:relative;
top:1px;
z-index:5;
height:20px;
vertical-align:top;
clear:both;
padding-top: 25px;
padding-right:25px;
padding-left: 5px;
}

/* The rest of the selectors for the ParentTabs id
define the elements within the ParentTabs div.
The tabs are made a fixed width here, but could be made to only
grow big enough to wrap around the text by setting width:100% */
```

```
#ParentTabs a {
float:left;
display:block;
width:75px;
text-align:center;
padding:5px 10px 4px 6px;
text-decoration:none;
font-weight:bold;
color:#f1f1ff;
background:url("TabRightLevel0.gif") no-repeat right top;
}

#ParentTabs a:hover {
text-decoration:none;
background-position:100% -150px;
color:#FFFFFF;
}


#ParentTabs li {
background:url("TabLeftLevel0.gif") no-repeat left top;
border-bottom: black 1px solid;
}

#ParentTabs li:hover {
background-position:0 -150px;
}

#ParentTabs .SelectedTab {
background-position:0 -150px;
border-width:0;
}
#ParentTabs .SelectedTab a {
background-position:100% -150px;
padding-bottom:5px;
color:#FFFFFF;
}

/*The container tab for the
Child Tabs is mostly used for positioning */

#ChildTabsContainer {
background-color:#336699;
padding-top:5px;
padding-bottom:5px;
border-top: black 1px solid;
vertical-align:middle;
text-align:center;
height: 20px;
padding-left: 300px;
}

#ChildTabsLevel1 a {
padding-left:5px;
padding-right:5px;
text-decoration:none;
```

```
font-weight:bold;
/* font-size:80%; */
color:#CCCCCC;
}

#ChildTabsLevel1 .SelectedTab a {
color:#FFFFFF;
}

#ChildTabsLevel1 a:hover {
color:#FFFFFF;
text-decoration:none;
}

#ChildTabsLevel1 .LinkLeft {
position:relative;
float:left;
height:20px;
border: 1px ;
padding-right:1px;
padding-left:1px;
background:url("BreadCrumbPipe.gif") no-repeat left bottom;
}
```

Creating a Template File for the template NavType

The template is very powerful, allowing you to configure most every aspect of the output of the NavMenu. As always, with that flexibility you also get some complexity. To understand the template file we'll break an example down piece by piece. At the high level, a template is made up of the **organizational/layout** pieces (a grouping so that parent-child relationships can be maintained), the **dynamic pieces** (pieces that have replacements made depending on the specific settings of a tab), and the **static pieces** (pieces that are transferred directly). The download package includes a sample skin called **SilverTabHover** that includes two template files; Snapsis.NavMenu.Parent.template & Snapsis.NavMenu.Child.template. We will use these templates as guides to describing how the templates work.

The organizational / layout tags are made up of tags/tokens to tell the parsing and rendering engine of the NavMenu what html markup you want to use. Each tag is created with square brackets in a way to make it easy to see the actual html code that is used. For example:

[NavMenu-Container[put opening html for the NavMenu container here **]]**

In the tag above we are capturing the html that will be used in the opening container between the inner square brackets. So the actual tag in the template file for this piece may look like this (you can use any html that you need to create your menu):

[NavMenu-Container[<ul class="Level0"> **]]**

To create a true hierarchy the parent elements need a way to enclose or "contain" child elements. The closing tags of the containers are created by specifying the same tag preceded by a forward slash. So the closing tag of the NavMenu-Container described above is:

[/NavMenu-Container[**]]**

Within the template, we have layout tags to signify the overall NavMenu-Container (as described above) and the following tags for the rest of the items (where ***n*** is the level number starting at 0 for root):

- [Level***n***-Container[ ]]
  - [Level***n***-Item[ ]]
    - [Level***n***-Anchor[/]] ← the anchor is a self closing tag
  - [/Level***n***-Item[ ]]
- [/Level***n***-Container[ ]]

Html can be added both above and below the template code.
If you place the html in between head tags like the following example they will automatically be moved up to the head section of the current page.

```
<head>
    <link id="silk" rel="stylesheet" type="text/css" href="menu.css" />
    <script language="javascript">
        function NavMouseover(li) {
            li.className+=' hover ';
        }
        function NavMouseout(li) {
            li.className=li.className.replace(' hover ', '');
        }
    </script>
</head>
```

Dynamic tokens are signified by using curly braces {token}
Inside the html above we include a dynamic token that will be replaced by the ID of the NavMenu when this template is parsed before rendering for example: {Page.Name}.

The current list of Dynamic tokens are:

- **SkinPath** = the absolute path to the skin folder (same as <%=SkinPath%> in a skin)
- **NavMenu.Id** = the ID of the menu as defined in the settings of the .ascx skin file.
- **Page.PageName** = the name of the page as defined in page settings of your portal.
- **Page.Title** = the title of the page as defined in page settings of your portal.
- **Page.Url** = the Url of the page (link item).
- **Page.Level** = the Level of the page in the parent-child hierarchy.
- **Page.Keywords** = the keywords of the page in as set in page settings.
- **Page.IconFile** = the IconFile as defined in page settings (referenced relative to the skin folder).
- **Page.IsActive** = True or False depending on if the page or one of its ascendants is currently active.
- **Page.IsAdminTab** = True or False depending on if the page is an admin page.
- **Page.IsDeleted** = True or False depending on if the page has been deleted (and is still in the Recycle Bin).
- **Page.DisableLink** = True or False depending on the disabled setting in page settings.
- **Page.IsVisible** = True or False depending on the hidden setting in page settings.
- **Page.Parent.ID** = The id of the parent to this page.
- **Page.HasChildren** = True or False depending on if the page is a parent to other page(s).
- **Page.Parent** = The complete Parent tab if page is a child. This token can then be used to access all the properties of the parent page as defined above. For Example Page.Parent.Keywords, etc.

IIF (condition, true part, false part) `false part is optional`

You will also notice that some of these dynamic tokens produce a logical True / False output which is not all that beneficial unless you have a way to evaluate the true/false in a condition. For that we have the classic IIF function. For purposes that are also beneficial in rendering the different navigation systems we have also included another replacement token:

- **Browser =** the current browser type requesting the page **(IE, Mozilla)**

With this function and token you can create output specific to certain browsers:

IIF({Browser}='IE', 'put IE specific output here', 'put non-IE output here')

Or if you only need output for IE then you can write it without the optional false part:

IIF({Browser}='IE', 'put IE specific output here')

Static Strings - [$ {token} [*text / html / javascript*]$]

Static strings allow you a way to create your own replaceable tokens. They are mainly used to make the template more readable so that you don't have to look at the same static string over and over again.

To create a static string you use the bracket nomenclature described for organizational tokens and a dollar sign (**$**) to signify that it is a string:

[${IEHoverJS}[** onmouseover="this.className+=' hover ';" onmouseout="this.className=this.className.replace(' hover ', '');" **]$]

The replacement token **{IEHoverJS}** can now be used anywhere you want to replace it with the string that was defined inside the tag.