300-6400 Roberts Street | Burnaby BC | V5G 4C9

Design Guide

Active Content Manager Version 8.0

Last revised September 11, 2006

The Active Network, Ltd.
Third Floor, Escom Building
6400 Roberts Street
Burnaby, BC
Canada V5G 4C9
Office telephone: 604.438.7361 or 1.800.661.1196
Fax: 1.604.432.9708

Contents

Overview

This guide provides a comprehensive introduction and detailed instructions for working with the Active Content Manager Design Manager and template system.

This guide is divided into different sections covering the Design Manager, the Design Package, setting up the Dynamic Menus, Editor configurations, as well as the IronPoint tags and Pagetypes.

Requirements: This guide is geared towards those with a solid understanding of HTML and CSS, and who have some familiarity with XML, and basic introductory training to the Active Content Manager.

What's New in the Design Guide?

There is no new functionality in the 8.0 release that affects the Design Guide.

Active CM and Design

The Active Content Manager allows users near-limitless possibilities when styling their websites. It employs a Design Package, which is a set of HTML, CSS and XML files, to allow users to design and configure the templates and dynamically generated menus on their site.

The Active CM also employs different 'Page Types' to enable users to build pages with their own content in a WYSIWYG or HTML Editor, as well as have dynamic content generated for them.

Definition of a Template

A template is a basic HTML framework in which content from the Active CM is organized and displayed. When creating a template, the user can decide what elements to display statically, and what areas they will display specific content dynamically with special 'IronPoint Tags'.

A template can be designed to ensure that a consistent look is achieved throughout the site, or several templates can be used to apply different looks to different areas of the site.

Every page has a template assigned to it and each template can hold different kinds of editable page content, including page type, syndicated pagelets, dynamic menus, etc.

Templates within the Active CM are managed in the Site Design system tool. The Site Design system tool performs two functions:

- Allows user to upload, manage, and version templates
- Controls the assignment of templates to pages

The Site Design Manager

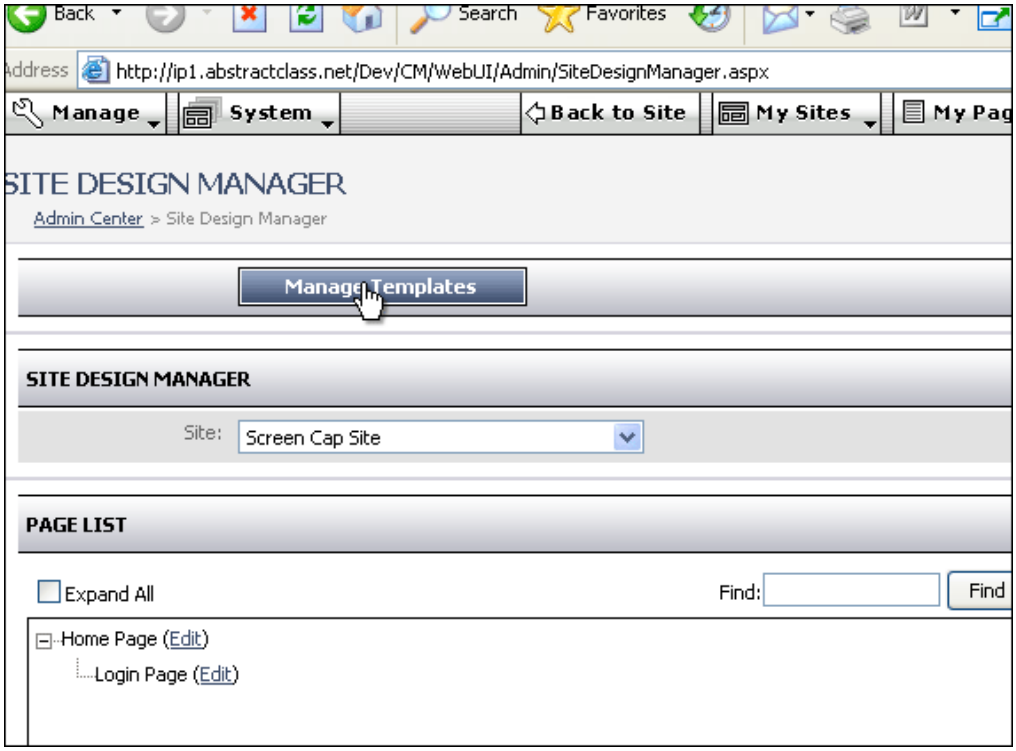
In the Active Content Manager, the Site Design Manager is where you upload and download design packages, and apply templates to different pages on your site. A new site will have the Default Design template already applied to it.

To access the Site Design Manager:

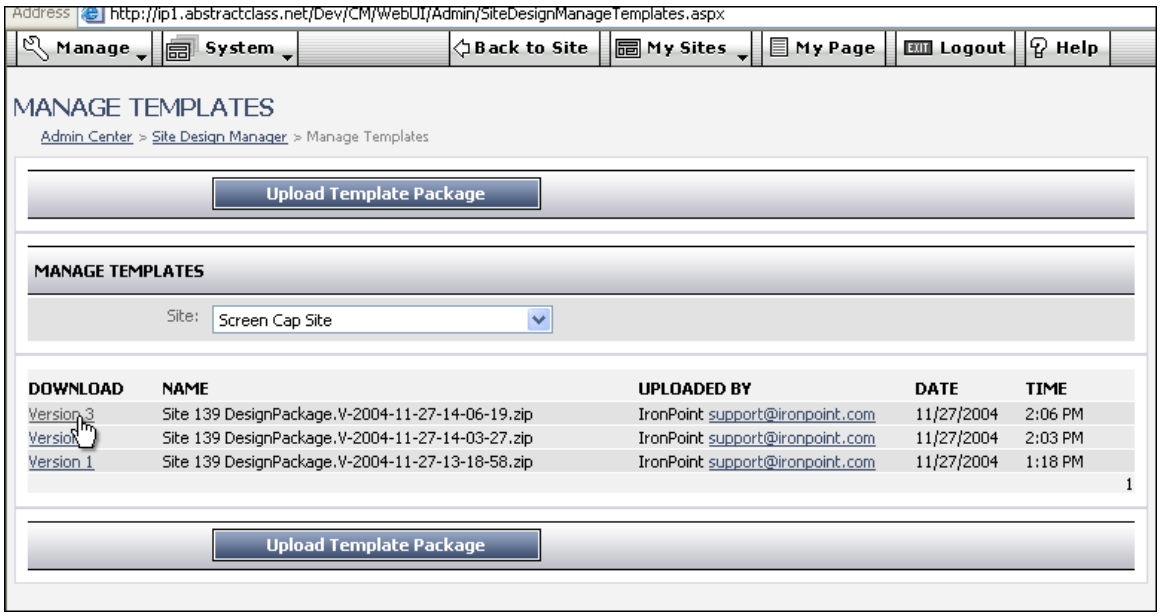
1. Login to your site. Once you're logged in, you will have access to the Application Toolbar.
2. Click on 'Admin Center' in the Toolbar.
3. From the System tools select Site Design, if the Site Design option is not visible contact your System Administrator.

   You are now in the Site Design Manager. This is where you access the Template Manager to download and modify the Design Package, which contains the site's template files. You are also provided with a Page List of your site where you view and assign design elements to the pages of your site.

4. To download the Design Package, click on Manage Templates.



5. To download the current Design Package, click on the first 'Version' link in the list; you can also download a previous version and revert to it by uploading it again.



6. In the window that pops up, select **Save**.
7. Select the desired location and save. It's best to create a new folder to save your design package in. You can save the design package in any location and with any name you like.

The Design Package

In the Active Content Manager, a Design Package is a zipped folder that contains the files that define and control your website's templates.

When a new site is created, the default template is applied to it. If you have multiple sites running from the same instance of the application, select the site you'd like to work on from the Site dropdown list. Each site will have a list of uploaded Design Packages, numbered as Versions. When you upload a Design Package it is versioned, allowing you to track when it was uploaded and by whom. Versioning allows you to roll back to previous versions. For a new site, there will only be Version 1, which is the Default Design Package.

The Design Package's name is generated by the Active CM and includes Site ID, and upload date and time. Regardless what you name your design package before uploading it, it will be renamed in this fashion once uploaded to the system.

To work with a Design Package:

1. Open the folder where you saved the zip file, unzip the file.
2. A folder named 'Templates' will be unzipped. Open the folder, inside you will find the files and images that make up a design package.

Anatomy of a Design Package

DefaultDesignTemplate.htm

A basic HTML Template used in the default Design Package. The template includes some styling, images, and basic IronPoint tags.

PageletDesignTemplate.htm

This is the template that Search Results and QuickPoll Results are automatically returned in. Unlike regular templates, it does not need to be listed in the SiteDesignManifest.xml, and is not applied to pages.

Panelbar.xml

The configuration file used for the Accordion-style menu used in the Active CM. Although not used in the Default Design Template.

SectionMenuData.xml

The configuration file used for the Flyout menu used in the Active CM.

SiteDesignManifest.xml

The configuration file used to name and organize the templates used in your design.

CSS Folder

DefaultDesignTemplate.css

This is the CSS used to style elements of the DefaultDesignTemplate.htm file.

HTMLEditor.css

This is the CSS used to style the content within the WYSIWYG editor without affecting any code outside of it. This enables the editor to display your content correctly within your template, or separate from it, and allows for separation between template and content styling. As a default, the CSS classes in this file are also used to populate the CSS Dropdown in the HTML Editor.

This file is explained in greater detail in the WYSIWYG Editor section.

Pagetypes.css

The Active CM dynamically writes the HTML content of various Page Types, including Site Maps, Calendars and Surveys. The PageTypes.css styles all the classes applied to these different pages. PageTypes and their styling is explained in greater detail in the Page Types section.

Panelbar.css, SectionMenuStyle.css and UnorderedListMenu.css

These are the CSS files that are used to design the three different kinds of dynamically generated menus (primary navigation) generated by the Active CM. They are explained at greater detail in the Section Menu section.

ip-surveyemail.css

The Active CM uses these styles for email sent when a survey is submitted. The default version is kept within the system folder, if you want these emails styled differently on a per site basis, change the styles and add an ip-surveyemail.css file to the design package.

Modalbutton.gif

This image file is used as a background for buttons on Survey pagetypes as a default. It is tucked away in this folder to ensure that it isn't inadvertently deleted.

Config Folder

HTMLEditor.config

File used to configure the HTML Editor in the Active CM; it is used to size the editor, choose the content in the editor's dropdowns, etc.

You can also create a unique HTMLEditor.config for each template by including the name of the template at the beginning of the HTMLEditor.css file. To create an HTMLEditor.config file to be used only with the template SubpageTemplate.htm, you would create another .config file named SubpageTemplate_HTMLEditor.config.

If a template doesn't have a specific .config file, it will use HTMLEditor.config as a default.

Images Folder

Miscellaneous images used in the Templates.

Templates and IronPoint Tags

Templates are created using a combination of HTML and 'IronPoint Tags', which are used to pull different kinds of content from the Active CM. IronPoint Tags are used to display items like Page Title, Location Line, Navigation Menus and page conent. When viewing a template in a browser, the names of the tags appear, but when the template is applied to a page on the ACM, the tags will be replaced with content.

For example, when opened in a browser, the Default Design Template it looks like this:

The HTML from the Default Design template is displayed below, stripped down to exclude some elements that are purely design to focus on the tags:

```
01 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
02 <html>
03 <head>
04 <title>IronPoint Default Vertical Template</title>
05 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
06 <link href="css/DefaultDesignTemplate.css" rel="stylesheet"
          type="text/css">
07 <link href="css/SectionMenuStyle.css" rel="stylesheet" type="text/css">
08 <link href="css/PageTypes.css" rel="stylesheet" type="text/css">
09 </head>
10 <body>
11 <div align="center">
12 <table class="layout" id="vertical" cellpadding="0" cellspacing="0">
13 <tr><td class="logo" colspan="2" valign="bottom"><img
          src="images/company_name.gif"></td></tr>
14 <tr><td colspan="2" class="greenstripe"></td></tr>
15 <tr><td valign="top" class="sectionmenu">
16   <ironpoint>SectionMenu</ironpoint></td>
17   <td valign="top" class="content">
18   <ironpoint>PageTitle</ironpoint>
```

```
19   <ironpoint>LocationLine</ironpoint>
20   <ironpoint>PageType</ironpoint>
21   <ironpoint>ChildLinks</ironpoint></td></tr>
22 <tr><td colspan="2" class="footer"></td></tr></table>
23 </body>
24 </html>
```

The IronPoint tags are bolded to help them stand out; their jobs are as follows:

<ironpoint>sectionmenu</ironpoint> dynamically displays the site's Section Menu, the navigation generated by the Active CM.

<ironpoint>PageTitle<ironpoint> displays the title of the page.

<ironpoint>LocationLine</ironpoint> displays a navigational 'breadcrumb trail' for the page.

<ironpoint>PageType<ironpoint> displays the main content of the page.

<ironpoint>ChildLinks<ironpoint> displays an unordered list of the child pages, or the pages that appear 'below' the current page in the structure of the site.

For a complete list of the IronPoint tags and their functions, please see Appendix I.

Also worth noting:

- The charset is set to utf-8; this is to ensure the characters display correctly when the dynamic page is published (Line 5).
- The linked CSS files. Along with the CSS used to style the template (DefaultDesignTemplate.css), there is also one linked that styles the menu (SectionMenuStyle.css) and another that styles various page elements generated by the Active CM, (pagetypes.css). The stylesheets will be covered in more detail later in this document (Lines 6 – 8).

SiteDesignManifest.xml

The SiteDesignManifest.xml is used to assign IDs to the different templates, which the Active CM will use to associate the templates to pages.

When you first download the Design Package, the SiteDesignManifest.xml will look a lot like this:

```
<Templates>
   <Template>
       <ID>2</ID>
       <Name>Default Design Template </Name>
       <FileName>DefaultDesignTemplate.htm</FileName>
   </Template>
</Templates>
```

The **<Templates>** tags are the root level; everything from **<template>** tag to **</template>** tag (including the tags themselves) is information that is needed for

each template. The ID is a number generated by the application as a unique identifier for the template; the Name is what appears in the Site Design Manager dropdown . It, too, needs to be unique, to prevent confusion for anyone applying the template to pages of the site later on. The FileName is the name of the template's HTML file.

Note The name that a template is assigned using the **<Name></Name>** tags needs to be **unique across all sites within a single instance of the ACM**. This means you can not have one site with the template name, "About Us" and a second site with the template name, "About Us". If template names are duplicated within a single instance of the software an error will be generated when you attempt to upload the design package.

When adding another template to the design package, the final step is to add it to the SiteDesignManifest.xml. To do this, copy the information from **<template>** to **</template>**, change the name and filename appropriately, and remove the ID. A new ID will be assigned to the template when the Design Package is uploaded.

If you added a template called "SubpageTemplate.htm" to your design package, your updated SiteDesignManifest.xml would look like this:

```
<Templates>
   <Template>
       <ID>2</ID>
       <Name>Default Design Template </Name>
       <FileName>DefaultDesignTemplate.htm</FileName>
   </Template>
   <Template>
       <ID></ID>
       <Name>SubPage Template</Name>
       <FileName>SubpageTemplate.htm</FileName>
   </Template>
</Templates>
```

You can add multiple templates to the SiteDesignManifest.xml at one time.

New Templates

To add a new template to your Design Package:

1. Create your new design template and save it in the Templates folder. If you are also creating a new CSS file, add it to the CSS folder; all images should be saved into the Images folder. This will help keep your design folder organized.
2. Edit the SiteDesignManifest.xml file. Adding a new template to the SiteDesignManifest.xml file is explained in the Site Design Manifest section.

PageletDesignTemplate.htm and PrintDesignTemplate.htm

Unlike regular templates, the PageletDesignTemplate and the PrintDesignTemplate are used under specific circumstances by the application, and do not need to be listed in the SiteDesignManifest.xml.

The PageletDesignTemplate is automatically applied to 'Pagelet Results', such as Quickpoll results, or Search results. You can change the Pagelet Design Template to look any way you like, and as long as it is named 'PageletDesignTemplate.htm', the Active CM will automatically apply it to the appropriate pages.

The PrintDesignTemplate is similar to the PageletDesignTemplate in the way it behaves, but it has a different function: it enables you to create a less graphics-intensive template to be applied to your pages for printing.

To use the PrintDesignTemplate:

1. Create the template 'printdesigntemplate.htm' for your design package. You can style it any way you like (keeping in mind that the purpose of the template is to not be as graphic intensive as your site templates).
2. On the other site templates add the IronPoint print template tag **`<ironpoint>printpage</ironpoint>.`** When the template package is uploaded and applied to a site, the PrintPage tags will become a link that will open the page in a new window using the print template.

Dynamic Menus

The menus used in the Active Content Manager are dynamically generated by the content in your site – as pages are added to the website, they are automatically included in the menu, preventing the need to update the navigation every time a new page is added.

There are three different menu types used in the Active CM:

- ASP.NET Menu from ComponentArt (www.componentart.com),
- r.a.d. Panelbar menu from Telerik (www.telerik.com) and
- Unordered List menu that was based on various articles from HTML Dog (http://www.htmldog.com/articles/suckerfish/dropdowns/) and A List Apart (http://www.alistapart.com/articles/dropdowns/).

The ASP.NET and RAD Panelbar menus are styled and controlled using XML and CSS, and the Unordered List menu is styled and formatted using CSS and JavaScript.

The menus are placed in Templates with the **`<ironpoint>sectionmenu</ironpoint>`** tags. Different attributes can be set for the menus (See the Ironpoint Tags section for more information).

The menus are styled using a separate CSS file, and are configured using an XML file. When the menus are rendered on your site, they are created as nested tables with different CSS classes applied to different levels, as specified in your menu XML file.

Depending on the menu's design, you may need to construct detailed xml configuration files or very simple ones. The xml files enable you to specify width, orientation, images, icons, pageID's and styling for your menu items.

ASP.Net Menu

This flyout menu type can be used for any menu that has the childpage links appear in a smaller 'flyout' menu, from the parent link when moused over.

Note The element itself is called the SectionMenu. Because it is the default, the Flyout is often referred to as the SectionMenu, and its XML and CSS files are named after it; SectionMenuData.xml, and SectionMenuStyle.css, respectively

The flyout menu is the default menu type for the **`<ironpoint>sectionmenu</ironpoint>`** tag, and it also uses the tag's default XML file: "SectionMenuData.xml." To specify these properties, you would write the tag like this:

```
<ironpoint FileName="SectionMenuData.xml"
MenuClass="ASPNetMenuFlyOut">sectionmenu</ironpoint>
```

Note When setting the attributes in the IronPoint tags, the case of the letters is important – make sure to duplicate the case of the attribute value pairs.

The Section Menu is configured with the SectionMenuData.xml file. The SectionMenuData.xml file uses a menu control to render the menu. The menu control can render images, text, and dynamic HTML menuing based on XML menu structure.

The SectionMenuData.xml file:

Here is an example of how a very basic SectionMenuData.xml file could look:

```
<SiteMap
ImagesBaseURL=”images/”
Orientation=”Vertical”
DefaultItemCssClass=”ClassName”
DefaultItemCssClassOver=”ClassNameOver”
>
</SiteMap>
```

A more advanced SectionMenuData.xml file might look like this:

```
<SiteMap
ImagesBaseURL=”images/”
DefaultGroupCssClass=”groupclass”
Orientation=”Vertical”
DefaultItemTextWrap=”True”
DefaultItemWidth=”200px”
DefaultItemCssClass=”classname”
DefaultItemCssClassOver=”hoverclassname”
DefaultCssLevel3=”levelthree_classname”
DefaultCssOverLevel3=”levelthree_hoverclassname”
>
     <Item
           ID=”PageID_#”
           Look-ImageUrl=”image.gif”
           Look-HoverImageUrl=”imageover.gif”
           Look-ImageHeight=”#”
           Look-ImageWidth=”#”
           DefaultSubGroupCssClass=”class”
           DefaultSubItemLook-CssClass=”subclass”
       DefaultSubItemLook-ImageUrl=”image.gif”
       DefaultSubItemLook-HoverCssClass=”hoversubclass”
           >
     <Item>
           <Item />
     </Item>
</Item>
</SiteMap>
```

This SectionMenuData.xml file sets different CSS classes for different levels of the menu, as well as different CSS classes for hover states, etc. Most menus don’t need a complex XML configuration file, but it enables you to have tighter control over the different elements of the menu if you choose to.

For a complete lists of attributes available for the SectionMenuData.xml, please see Appendix III.

SectionMenuStyle.css

The styles for the Flyout SectionMenu come from the SectionMenuStyle.css file.

The class names are rendered with the menu HTML as set in the SectionMenuData.xml file. The SectionMenuData.xml is a way to assign CSS classes to HTML that is dynamically generated. The SectionMenuStyle.css is used to set the styling for the classes used in the SectionMenuData.xml – it isn't mandatory to keep the menu styles in a separate CSS file, but it certainly helps keep things organized.

Flyout Menu Examples

Example 1: The Default Design Template menu

The Default Design that comes with the Active CM employs the ASP.Net flyout menu. Once your site has a few more links, it will start to look like this:

Here is the SectionMenuData.xml file for this menu:

```
01        <SiteMap
02              DefaultItemCssClass="ipf-
                SectionMenuItem"
03              DefaultItemCssClassOver="ipf-
                SectionMenuItemOver"
04              DefaultCssLevel1="LevelOne"
05              DefaultCssOverLevel1="LevelOneOver"
06              Orientation="vertical"
07              DefaultItemWidth="190px"
08              DefaultItemTextWrap="True"
09              >
10        </SiteMap>
```

The properties being set with the XML file are as follows:

Line 1: Beginning of opening SiteMap tag.

Line 2: The default CSS class for the menu is set.

Line 3: The default hover CSS class for the menu is set – as with regular links, this styles how the menu will look on mouseover.

Line 4: Sets the CSS for the first level of the menu to be different than the default CSS applied to all other levels. In this case, it gives each item a dashed white border on the bottom.

Line 5: Sets the hover CSS for the first level of the menu to be different than the default CSS applied to all other levels.

Line 6: Sets the Orientation of the menu; the ASP.Net default is horizontal.

Line 7: Sets the width for each menu item.

Line 8: Specifies whether or not the text in the menuitems should wrap – the default setting is False. This overrides the set width of the item, so if you set your width, but have not specified that you want the text of the items to wrap, menu items with longer names with force the menus wider.

Line 9: End of opening SiteMap tag.

Line 10: Closing SiteMap tag. As this is XML, and there is nothing between the opening and closing SiteMap tags, you can also end the opening SiteMap with '/>', and then you won't need to add the closing SiteMap tag.

This is the SectionMenuStyle.css for this menu:

```
01    .LevelOne
                {background: #ECEDEC;color:#39526A;cursor: hand;      }

02    .LevelOneOver
                {background: #E1E2E1;color:#39526A;cursor: hand;      }

03    td.LevelOne,
            table.LevelOne td table td,
            td.LevelOneOver,
            table.LevelOneOver td table td
                  {padding: 10px 20px; font-size:11px;
                  border-bottom: 1px dotted #ffffff;}

04    .ipf-SectionMenuItem
                {background: #ECEDEC;color:#39526A;cursor: hand;}

05    .ipf-SectionMenuItemOver
                {background: #E1E2E1;color:#39526A;cursor: hand;}

06          td.ipf-SectionMenuItem,
            table.ipf-SectionMenuItem td,
            td.ipf-SectionMenuItemOver,
            table.ipf-SectionMenuItemOver td
                  {padding: 10px 20px; font-size:11px;}
```

Most of the contents of the SectionMenuStyle.css are pretty self-explanatory to anyone familiar with CSS – the only part that might require explanation is Lines 3 and 6.

It will probably help to see what the system is generating when it's building the menus:

```
01        <table>
02        <tr>
03              <td class="ClassName"></td>
04        </tr>
05        <tr>
06              <td>
07                    <table class="ClassName">
08                          <tr><td></td></tr>
09                    </table>
10              </td>
11        </tr>
12        </table>
```

Lines 02 – 04: a basic Menu Item with no flyout. The classname is applied to the td that the menu Item appears in.

Lines 05 – 11: a Menu Item with a flyout. Instead of being applied to the td, the class is applied to the table inside of the td, and the flyout items are nested further in the table.

Therefore when adding padding to Menu Items, simply placing the padding on the class applied to a specific item has unpredictable results – more often than not, items with flyouts will appear to have double the padding as those that do not.

That is why, in the SectionMenuStyle.css, padding is applied to multiple, specific items, like in this example of the ipf-SectionMenuItem, which is applied as the Default Design CSS to all Menu Items:

```
td.ipf-SectionMenuItem,
table.ipf-SectionMenuItem td {padding: 10px 20px;}
```

The padding is applied specifically to td.ipf-SectionMenuItem, to add it to Menu Items without flyouts, as well as to table.ipf-SectionMenuItem td, to add it to td's *inside* of tables with the .ipf-SectionMenuItem class. This ensures that padding isn't doubled up in some table cells, and that the styling acts as one would expect.

The menuing also acts slightly different when a style is applied to a specific level, as opposed to all Items.

The rendered HTML for a menu with a style applied to a specific level looks something like this:

```
01        <table class="ClassName">
02              <tr>
03                    <td class="ClassName"></td>
04              </tr>
05              <tr>
```

```
06                        <td>
07                              <table>
08                                    <tr><td></td></tr>
09                              </table>
10                        </td>
11                  </tr>
12            </table>
```

The class to style a specified Level of the menu is actually applied to the very first table tag, as well as the td of a Menu Item without a flyout, and not at all to a Menu Item with a flyout. Therefore, to properly add styling such as padding or borders, the CSS needs to be written as follows:

```
td.LevelOne,
table.LevelOne td table td
            {padding: 10px 20px; border-bottom: 1px dotted #ffffff;}
```

Addressing Menu Items without flyouts is done the same as it is in the CSS styling default menu items. However, to style menu items *with* flyouts, they need to be addressed from the top level table downwards.

Example 2: Images as Top Level Menu Items

Instead of displaying the text generated by the Active CM,

And the menu you would like to build might also be horizontal:

The SectionMenuData.xml for this menu would look like this:

```
01          <SiteMap
02                DefaultItemCssClass="ipf-SectionMenuItem"
03                DefaultItemCssClassOver="ipf-SectionMenuItemOver"
04                ImagesBaseURL="Images/"
05                Orientation="Horizontal"
06          >
07                <Item ID="PageID_6" Look-ImageUrl="menulogin.gif"
                        Look-HoverImageUrl="menuloginover.gif"/>
08                <Item ID="PageID_11" Look-ImageUrl="menu01.gif"
                        Look-HoverImageUrl="menu01over.gif"/>
09                <Item ID="PageID_12" Look-ImageUrl="menu02.gif"
                        Look-HoverImageUrl="menu02over.gif"/>
10                <Item ID="PageID_13" Look-ImageUrl="menu03.gif"
                        Look-HoverImageUrl="menu03over.gif"/>
11                <Item ID="PageID_14" Look-ImageUrl="menu04.gif"
                        Look-HoverImageUrl="menu04over.gif"/>
12                <Item ID="PageID_15" Look-ImageUrl="menu05.gif"
                        Look-HoverImageUrl="menu05over.gif"/>
13          </SiteMap>
```

Line 1: Start of opening SiteMap tag

Line 2: Sets default CSS class for Menu Items

Line 3: Sets default hover CSS class for Menu Items

Line 4: Sets the location of images used in the menu – this way, you don't need to specify the file location of images as you use them in the menu, you just need to specify their names.

Line 5: Sets orientation of first level menu items.

Line 6: End of opening SiteMap tag

Line 7 to Line 12: Individual Menu Items with the following properties:

ID – Sets the Page ID for the Item, pairing it with a specific page and all of that page's childpages.

Look-ImageUrl – Sets the Image used for this Menu Item

Look-HoverImageUrl – Sets the Image used for this Menu Item in Hover

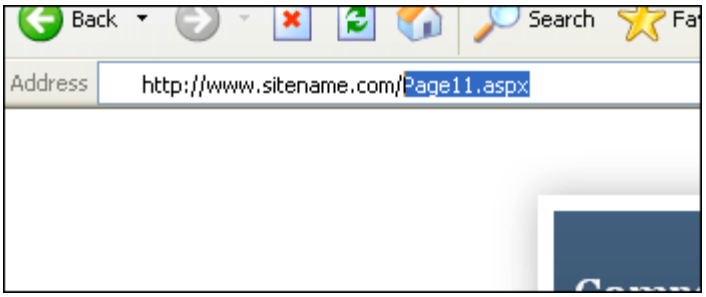
Line 13: Closing SiteMap tag.

Because you need to specify a page ID and specific image for every top-level item that you want to appear in the menu, this removes a bit of flexibility, as you cannot dynamically add another top level menu item without changing your XML file, and creating a new image.

However, adding another top level of navigation to a website is a pretty big change, and, in a well planned site, rarely necessary.

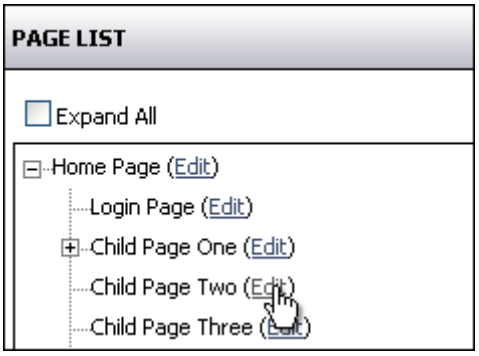
How to find the ID of a Page:

When on the page, the ID is part of the URL:

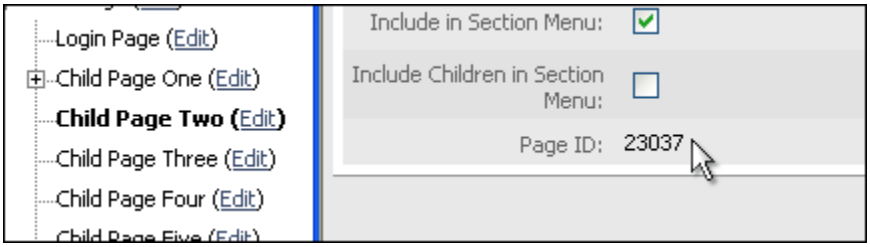


You can also find it in the Site Design Manager:

1. Click on the 'Admin Centre' Link in the Toolbar
2. Select 'Site Design' from the 'System' dropdown menu
3. Click the 'Edit' link next to the page you want the ID from:



4. This will open a window of the Page Design Details. The bottom line will state the Page's ID.



Accordion (AKA Panelbar menu)

The accordion, or panelbar menu, is a menu that expands to reveal childlinks of a page when you click on that link.

To place the accordion menu in an area of your template, you would type the following; in the case of the accordion menu, you need to specify the xml file being referenced, and the menutype.

```
<ironpoint FileName=”panelbar.xml” MenuClass=”
TelerikAccordion”>sectionmenu</ironpoint>
```

The Accordion menu is also configured with an xml file, though the file looks a little different than the one used for the flyout menu, as they use different technologies.

Here is an example of a panelbar.xml file:

```
<Menu
ImageDirectory=”/images”
DefaultMenuSpacing=”0”
DefaultLevelPadding=”0”
TableWidth=”190”
DefaultExpandEffect=”Stretch”
DefaultToolTip=”Click Here”
DefaultInitialState=”Collapsed”
DefaultCssLevel1=”nav-main”
DefaultCssLevel2=”nav-secondary”
DefaultCssLevel3=”nav-tertiary”
DefaultImageLevel1=”arrow.gif”>
    <Group>
          <Item>
                <Group>
                      <Item />
                </Group>
          </Item>
</Group>
</Menu>
```

In the panelbar.xml, the menu levels are created with the **<Menu>** tag as the base tag, and the menu items are created with nested **<Group>** and **<Item>** tags.

For a complete list of attributes available for the Accordion Menu, please see **Appendix IV**.

Accordion Menu Examples

This is an example of a basic Accordion Menu style:

The panelbar.xml for this menu looks like this:

```
01          <Menu
02          HeaderCss="headerCssClass"
03          HeaderSelectedCss="headerCssClassSelected"
04          DefaultItemCss="itemCssClass"
05          DefaultItemSelectedCss="itemCssClassSelected"
06          DefaultMenuSpacing="0"
07          TableWidth="190"
08          >
09          <Group></Group>
10          </Menu>
```

Line 1: Start of opening Menu tag.

Line 2: Sets the CSS class for the header, or top level, of the menu

Line 3: Sets the CSS for the selected header

Line 4: Sets the default item CSS
Line 5: Sets the default hover item CSS

Line 6: Sets the default spacing between top level menu items – not set, the menu items have 4px space between them

Line 7: Sets the width of the menu. Not set, the menu defaults to 200px wide.

Line 8: Closes the opening Menu tag.

Line 9: Group tags – child to the Menu tag, must be included to display menu.

Line 10: Closing Menu tag.

The panelbar.css for this menu looks like this:

```
01 .headerCssClass,
         .headerCssClassSelected ,
         .headerCssClass:visited,
         .headerCssClassSelected:visited
               {font-size: 11px; padding: 5px;font-weight: bold;
               color: #999999; text-decoration: none;}

02 table.headerCssClass,
         table.headerCssClassSelected
               {border-bottom: 1px solid #999999;}

03 .headerCssClassSelected,
         .headerCssClassSelected:visited
               {background-color: #9ABA77; color: #ffffff !Important }

04 .itemCssClass,
         .itemCssClass:hover,
         .itemCssClass:visited,
         .itemCssClassSelected,
         .itemCssClassSelected:hover
               {font-size: 10px; color: #658A3D;display: block;
               padding: 5px 0 5px 10px; background-color: #DAEBC7;
               text-decoration: none;  }

05 .itemCssClassSelected,
         .itemCssClassSelected:visited
               {color: #ffffff; background-color: #658A3D;}

06 .itemCssClass:hover,
         .itemCssClassSelected:hover
               {background-color: #ffffff;}
```

Line 1: Used on the header, or highest level, of navigation.

The HTML generated for a top level link with no childlinks looks like this:

```
<table class=”headerCssClass”>
      <tr>
      <td><a href="link" class='headerCssClass'>Childpage</a></td>
      </tr>
</table>
```

Because the class is applied to both the link itself and the table that holds, distinctions need to be made between what styles you’d like to apply to the table, and what you’d like to apply to the link. For example, with **Line 2**, we want a border on the bottom of the table, but not on the link, so we need to specify what to apply this style to.

Line 3: With the Accordion menu, classes are applied to <a> tags, so the hovers are styled with the pseudo-class :hover, instead of as a separate CSS class like with the

Flyout menus. However, the Accordion menu also has a Selected state, which is styled with this CSS class.

Line 4: CSS class applied to the subpage links. This class is only applied to the <a> tag; rendered HTML for a Menu Item with subpage links looks something like this:

```
<table class=”headerCssClass”>
  <tr>
    <td><a href="link"
         class='headerCssClass'>Childpage</a></td>
  </tr>
</table>
<table>
  <tr>
    <td>
      <table>
        <tr>
          <td><a href="link"
                          class='itemCssClass'>Grandchild</a></td>
        </tr>
      </table>
     </td>
  </tr>
  <tr>
    <td>
      <table>
        <tr>
          <td><a href="link"
                          class='itemCssClass'>Grandchild </a></td>
        </tr>
     </table>
   </td>
  </tr>
</table>
```

Line 5 and 6: CSS class applied to Selected and hover states of the subpage links.

To better understand where what classes are applied in the Accordion menu, you can view the source of the webpage after uploading your design. Unlike the ASP.NET menu, which renders as an array, the Accordion menu renders as normal HTML.

Unordered List Section Menu

The Unordered List Section Menu for the Active CM looks and acts a lot like the standard Flyout (ASP.Net) menu. However, the code generated for this menu is quite different.

This menu uses a series of nested unordered lists to create the different menu items and their parent/child relationships. Some CSS and a smattering of JavaScript are then used to turn the static lists into dynamic flyout menus.

The benefits of this menu are:

It uses semantic markup. Menus are generally considered lists that are displayed in no particular order, AKA an unordered list.

It is accessible to different devices. This includes screen readers, mobile devices, etc.

The code is lightweight. The HTML is super-clean, and the CSS and JavaScript are just enough to get the flyouts working.

To use the Unordered List menu, use the sectionmenu tag in your template with the following attribute:

```
<ironpoint
 MenuClass=”UnorderedList”>sectionmenu</ironpoint>
```

Note With the Unordered List menu, the standard menu tag attributes, such as MaxLevels and RootPageDefId will work.

Here is an example of the HTML that’s generated by this menu:

```
<ul id="sectionMenuElementID_0" class="ipf-sectionmenu">
<li id="pageid159"><div><a href="/Page159.aspx">Login Page</a></div></li>
<li id="pageid160" class="ipf-parent"><div><a
    href="/Page160.aspx">Calendar</a></div>
    <ul>
    <li id="pageid162"><a href="/Page162.aspx">Calendar Event</a></li>
    <li id="pageid162"><a href="/Page162.aspx">Calendar Event</a></li>
    <li id="pageid162"><a href="/Page162.aspx">Calendar Event</a></li>
    </ul>
</li>
<li id="pageid161"><div><a href="/Page161.aspx">Survey</a></div></li>
<li id="pageid275"><div><a href="/Page275.aspx">Help Desk</a></div></li>
<li id="pageid306"><div><a href="/Page306.aspx">Blog</a></div></li>
<li id="pageid394"><div><a href="/Page394.aspx">Photo Gallery</a></div></li>
</ul>
```

Like the standard flyout menu and the accordion menu, the code is dynamically rendered, which means there is a limit to how much control there is over each individual menu item’s styling. Unlike the other two menus, however, the new unordered list menu only uses a CSS file for formatting, and doesn’t require the use of an XML file to configure different menu properties.

The styling for different levels of the menu can be done through using the classes that are dynamically generated in the HTML (a “ipf-parent” class for each <li> that has a nested <ul> in it and a “ipf-sectionmenu” class on the parent <ul> tag) and by using descendant selectors (you can learn more about descendant selectors here: http://www.w3.org/TR/REC-CSS2/selector.html#descendant-selectors).

UnorderedListMenu.css

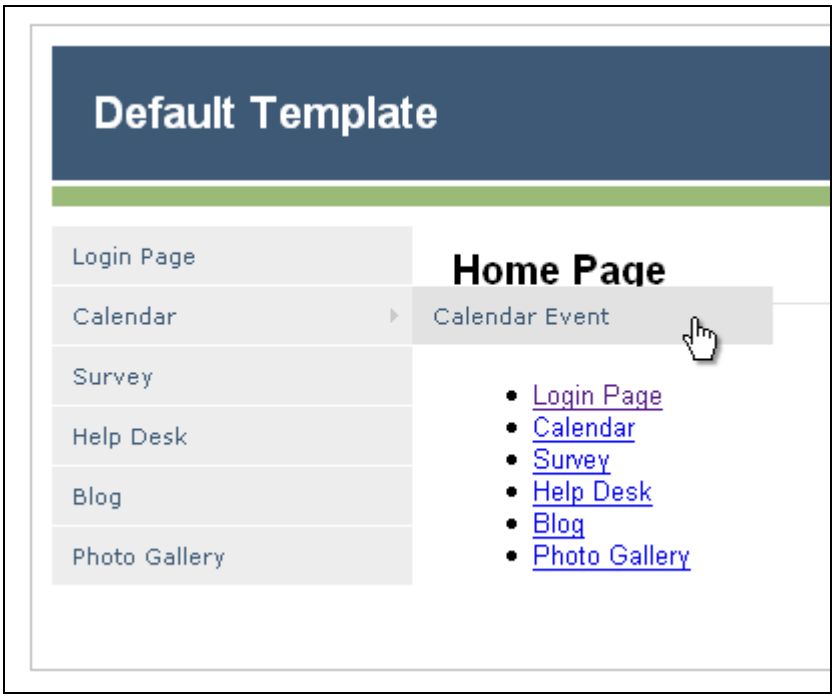
The styles and behavior for the Unordered List menu come from the UnorderedListMenu.css – without it, the menu just renders as standard nested unordered lists.

Note If you have upgraded to 7.6 without upgrading to 7.5, you can copy the CSS example from this document, or if you create a new site in the ACM, the new menu CSS will be included in the new site's template package. Any sites that were created after the 7.5 upgrade will include the default CSS for this menu in the default design template.

Like with the other menus, the CSS styling the Unordered List will need to be linked to the template that that menu is being used in. Similarly, the CSS doesn't need to reside in its own file – it entirely depends how you'd like your design package to be managed.

Unordered List Menu Example

Below is an example of the new Unordered List flyout menu styled to look similar to the standard ASP.Net Flyout menu.



As the styling is handled entirely in the CSS, there isn't any configuration done with an XML file – the styling is applied and handled solely through the CSS. Because of the nature of the menu, part of the CSS that's used for it deals with the actual look and feel of the menu, and the rest deals with the menu's behaviour (such as flyout positioning, etc):

```
01   .ipf-sectionmenu, .ipf-sectionmenu ul {padding: 0;margin: 0; list-style:
     none; float:left;}
02   .ipf-sectionmenu li {position:relative;   float:left;}
03   .ipf-sectionmenu li ul {position:absolute;left: -999em;margin-
     left:180px;top:0}
04   .ipf-sectionmenu li ul ul {left: -999em;}
05   .ipf-sectionmenu li a {width: 160px;display:block;color:#3d5975;text-
     decoration:none;background: #ecedec;border-bottom:1px solid
     #ffffff;padding:8px 10px}
```

```
06  .ipf-sectionmenu li.ipf-parent a{background: #ecedec
    url(../images/SectionMenuExpandImageRight.gif) right no-repeat;}
07  .ipf-sectionmenu li.ipf-parent a:hover{background: #e1e2e1
    url(../images/SectionMenuExpandImageRight.gif) right no-repeat;}
08  .ipf-sectionmenu li a:hover {background:#e1e2e1}
09  .ipf-sectionmenu li.ipf-parent li a{background-image:none;}
10  .ipf-sectionmenu li.ipf-parent li a:hover{background-image:none;}
11  .ipf-sectionmenu li:hover ul ul, .ipf-sectionmenu li:hover ul ul ul, .ipf-
    sectionmenu li.sfhover ul ul, .ipf-sectionmenu li.sfhover ul ul ul {left: -
    999em;}
12  .ipf-sectionmenu li:hover ul, .ipf-sectionmenu li li:hover ul, .ipf-
    sectionmenu li li li:hover ul, .ipf-sectionmenu li.sfhover ul, .ipf-
    sectionmenu li li.sfhover ul, .ipf-sectionmenu li li li.sfhover ul {left:
    auto;}
```

Line 1: Primarily removes list styling from the menu, and sets the menu's position in the template

Line 2: Sets positioning of the menu items relative to their parent menu.

Line 3: Sets position for the first level flyouts based on the menu's width.

Line 4: Hides the flyouts by setting a huge negative positioning for the flyout – this hides the content offscreen. This is done instead of 'display:none' because screenreaders will ignore items with 'display:none'.

Line 5: Actually styles the menu item by styling the link tag. In this case, the link is set to display block, and height, width, padding and colours are applied.

Line 6: Sets a background image for menu items with 'children' – in this case, an arrow is set as the background image so site visitors know that the menu item has a flyout.

Line 7: Sets the background of the menu items with children on hover.

Line 8: Sets the background for standard menu items on hover.

Line 9: As the menu in the example only shows one level of flyouts, we don't want that level to also display arrows if the pages have children as the third level isn't actually rendering in the menu. This line ensure that menu items on the second flyout with childpages will look the same as menu items on the second level that don't have childpages, as they will be acting the same.

Line 10: Same as above, but for the hover state of those items.

Line 11: This hides the flyout of menu items until they're hovered over.

Line 12: Corrects the left position set in the previous line; makes the flyout appear when the correct menu item is hovered over.

More information about the creation of, and the theory behind this kind of menuing can be found at the following links:

http://www.alistapart.com/articles/dropdowns/

http://www.htmldog.com/articles/suckerfish/dropdowns/

http://www.htmldog.com/articles/suckerfish/dropdowns/example/vertical.html

http://www.alistapart.com/articles/horizdropdowns/

WYISWYG Editor

The Active CM uses Telerik's RAD Editor (www.telerik.com) in the application to allow users who don't know HTML to add and edit content on the websites.

The HTMLEditor.config file and the HTMLEditor.css are the two files in the Design Packarge used to configure and style the WYSIWYG editor.

HTMLEditor.config File

The HTMLEditor.config file is used to add custom attributes to the HTML Editor. These include content for the dropdowns in the WYSIWYG editor, such as CSS classes, paragraph styles, code snippets and links; as well as the width and height of the editor itself.

The basic format of the HTMLEditor.config is standard XML, with the different settings listed between the start and end **<root>** tags.

The different configurations for the editor can been added in the root element:

<Config>

The **<editorwidth>** and **<editorheight>** children of the **<config>** tags let you to set the width and height of the editor in pixels.

You can also set whether you would like the CSS Dropdown in the editor populated by all CSS classes listed in the dropdown ('True') or if you would like to specify certain stylesheets for the dropdown to display instead ('False'). Please see CSS Class Dropdown for more information about linking specific files.

```
<config>
      <editorwidth>472</editorwidth>
      <editorheight>440</editorheight>
      <cssfromtemplate>false</cssfromtemplate>
</config>
```

<Links>

Between the parent **<links>** tags, the names and URLs of websites can be listed to enable users to add frequently used links at the click of a button. Each link listed needs a **<name>** and a **<value>**:

```
<links>
      <link>
          <name>Active Network</name>
          <value>http://www.theactivenetwork.com</value>
      </link>
      <link>
          <name>Your Company</name>
```

```
            <value>http://www.yourcompany.com</value>
        </link>
</links>
```

<Fonts>

Fonts can be added between the **<fonts>** tags for styling ease by using the following format:

```
<fonts>
        <font><name>Tahoma</name></font>
        <font><name>Verdana</name></font>
</fonts>
```

<Symbols>

Different symbols can also be added by using similar format as the fonts – this enables users to insert symbols like ©, ¶ or ® with ease. Please see Appendix V for a partial list of symbols available for the Editor.

```
<symbols>
        <symbol><name>\u20AC</name></symbol>
        <symbol><name>\u00A2</name></symbol>
        <symbol><name>\u00A3</name></symbol>
</symbols>
```

<CSSFiles>

If the "cssfromtemplate" value in the **<config>** tags has been set to False, the Editor will populate the CSS Class Dropdown from the HTMLEditor.css by default. By listing stylesheets in the <cssfiles> tags, you can override this default and pick specific stylesheets you would like displayed in the dropdown.

```
<cssfiles>
        <file><name>HTMLEditor.css</name></file>
</cssfiles>
```

<Paragraphs>

The Paragraph dropdown makes it possible to make specific HTML tags available to those using the WYSIWYG editor view. Each 'paragraph' needs values for both the **<name>** and **<value>** tag.

```
<paragraphs>
        <paragraph>
              <name>Heading 1</name>
              <value>&lt;H1&gt;</value>
```

```
        </paragraph>
        <paragraph>
              <name>Heading 2</name>
              <value>&lt;H2&gt;</value>
        </paragraph>
</paragraphs>
```

<Colors>

The **<colors>** tags let you set different colours for the ForeColor Dropdown. Each colour needs values for both the **<name>** and **<value>** tags.

```
<colors>
        <color>
              <name>Pumpkin Orange</name>
              <value>#cc6600</value>
        </color>
        <color>
              <name>Dark Gray</name>
              <value>#888888</value>
        </color>
</colors>
```

<Snippets>

The **<snippets>** tags allow you to enter frequently used bits of HTML, and allow content providers to insert it into the editor with a click of a button. Each **<snippet>** needs data for both the **<name>** and **<value>** tags, and must follow the below formatting:

```
<snippets>
        <snippet>
              <name>Snippet 1</name>
              <value><![CDATA[<b>S</b>ample snippet 1.]]>
              </value>
        </snippet>
        <snippet>
              <name>Snippet 2</name>
              <value><![CDATA[<b>S</b>ample snippet 2.]]>
              </value>
        </snippet>
</snippets>
```

Multiple HTMLEditor.config files

The HTMLEditor.config is the default config file, and is applied to all templates. However, you can create a different config file for each template – all you need to do is preface the name of the config file with the name of the template.

For example, to use a specific HTMLEditor.config file for a template called HomePage.htm, you would name the config file: HomePage_HTMLEditor.config. That config file would only be applied to pages using the HomePage.htm template.

HTMLEditor.css File

The HTMLEditor.css is used to style the content of the Editor - this stylesheet makes it possible for content providers to see how the styling they're applying will look when it's published on the actual webpage.

As a default, the classes in the HTMLEditor.css are also used to populate the CSS Dropdown in the WYSIWYG Editor. If you would like to use a different file to populate this dropdown, please see instructions how to select stylesheets in the HTMLEditor.config section.

Here is an example of an HTMLEditor.css file:

```
body,td,th{
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size:12px;
    color: #666666;}

a,  a:visited {color: #7AA54B;}
a:hover {color: #4C7B18}

h1, h2, h3, h4, h5, h6 {
     font-family: georgia, garamond; weight: normal}
h1 {font-size: 14px; color: #526F8B}
h2, h3, h4, h5, h6 {font-size: 12px; color: #6798C6}

.orangebold {color: #cc6600; font-weight: bold;}
.redunderline {color: #cc0000; text-decoration: underline;}
.greyuppercase {color: #999999; text-transform: uppercase;}
```

In this example, all the styles would appear in the Editor content, and the three classes at the bottom (.orangebold, .redunderline and.greyuppercase) will all appear in the CSS Dropdown as a default (please see the section on Using the HTMLEditor.config for how to choose different CSS files to populate this list).

The HTMLEditor.css, like the HTMLEditor.config file, is automatically included with your templates, and doesn't need to be linked to them. Also like the HTMLEditor.config, you can make unique HTMLEditor.css files for specific templates by naming them to match. For example: to use a specific HTMLEditor.css file with a template called SubpageDesign.htm, name it SubpageDesign_HTMLEditor.css.

Note In the HTMLEditor.css, always use absolute font sizing. You may choose to use percentages or ems to size the fonts you use in your site, but there are a couple factors that make it important to use absolute font sizing for the Edit modes:

1. In QuickEdit mode, both the template CSS and the HTMLEditor.css will be applied to the document. So if you've have p {font-size: 80%} in the former, and p {font-size: 80%} in the latter, both will be applied to the WYSIWYG editor, making the font extra small. You can omit the font-sizing from the HTMLEditor.css file all together, but this means that no font sizing will be applied in Full Edit mode.

2. In FullEdit mode, if there isn't an absolute font-size set for the WYSIWYG editor, the base font will be inherited from the styling applied to the application's backend elements. In other words, p {font-size: 80%} will inherit the 11px font-size set for the elements of the Active CM, making it 9px high as opposed to the 13px it would be in regular view mode.

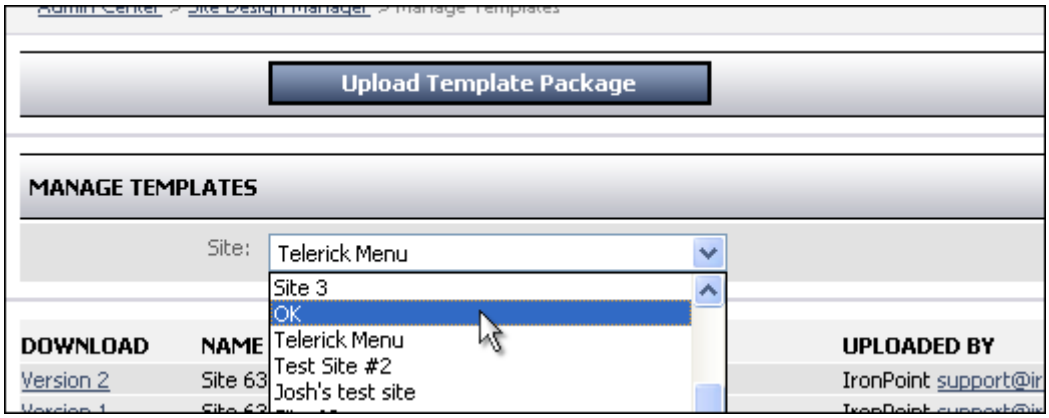
Uploading the Design Package

When you're done editing your design package, the next step is to upload it back to the site.

1. Compress your Design Package:
   Right click on your "Templates" folder
   Select "Send To" → "Compressed (zipped) Folder from the menu

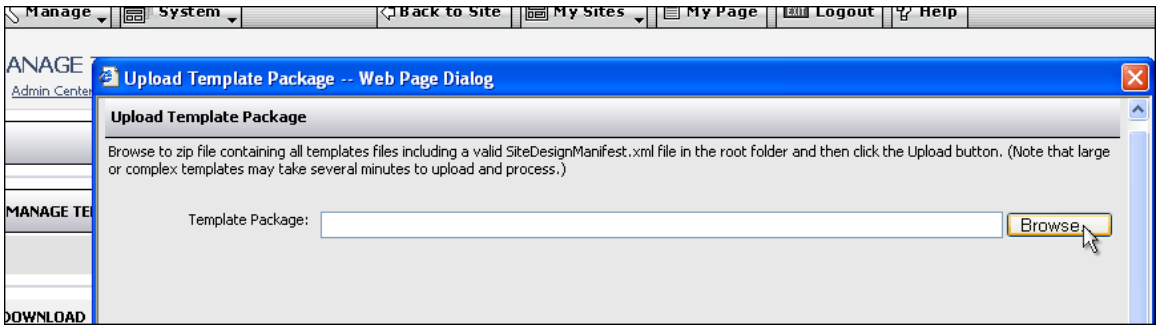
Note It's important that the zip file contains the same hierarchy as when it was originally downloaded – otherwise, the application will not be able to locate the files.

2. Return to the Manage Templates section of the Site Design Manager on your website.
3. Make sure the correct site has been selected in the site-dropdown box…



… then click the "Upload Template Package" button

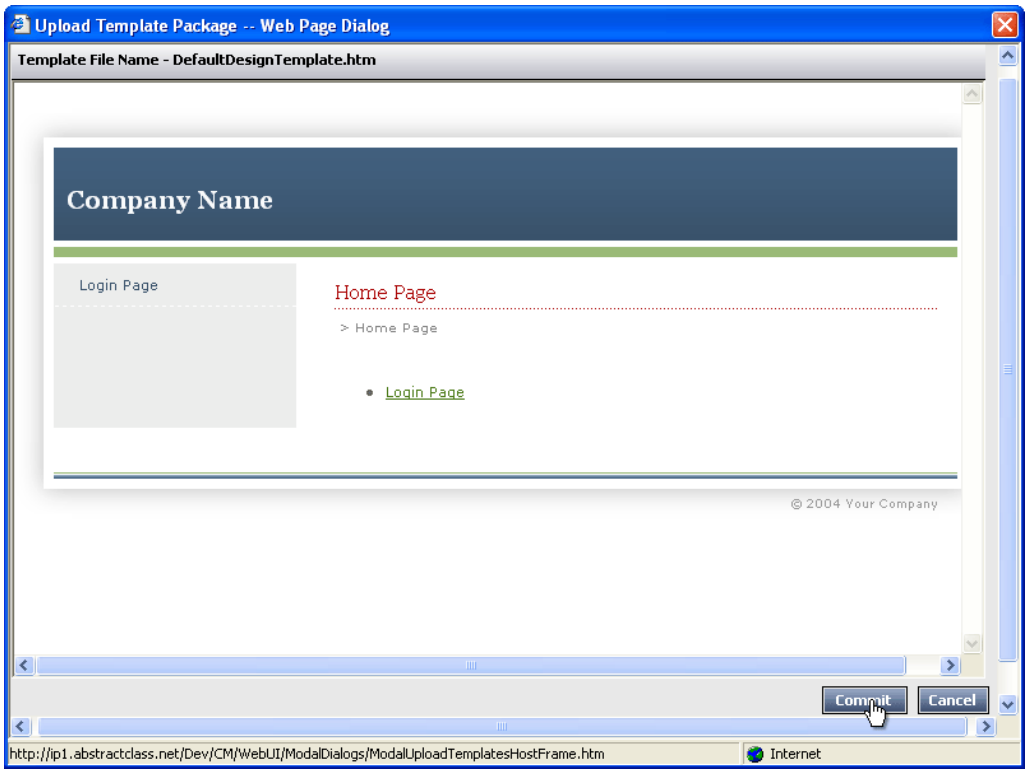
4. A pop-up window will open. From there, click the 'Browse' button to find your zipped Design Package. Select it, and click the "Upload" button.



Depending on how large the design package is and the speed of your internet connection, this could take several minutes.

Note The Design Package must be in .zip file format.

5. You will now be able to preview all of the templates in the design package. If there is a problem, click the cancel button; if not, click "Next" until you get through all of your templates.



6. Once you have finished previewing, you have the option of clicking the "Commit" button.

Note The new design package will appear with a new version number. All existing templates will be updated and all new templates will be available.

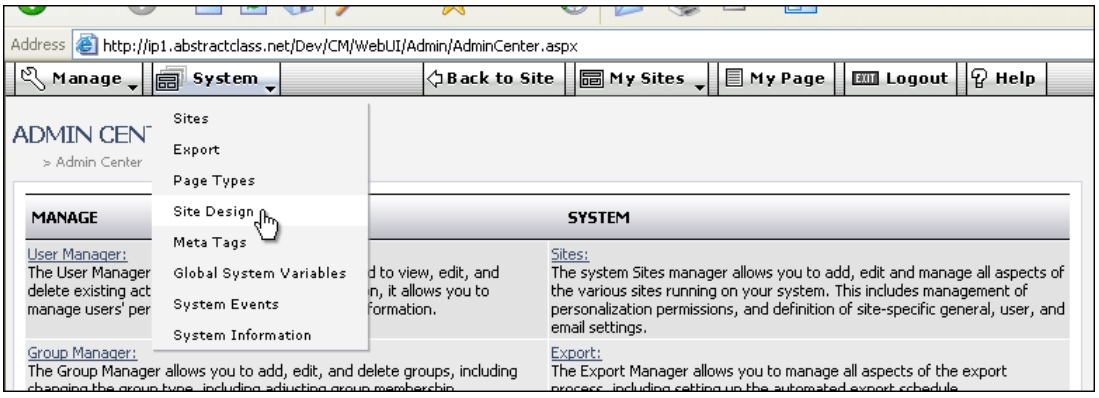
7. Your saved Design package will now be applied to the site.

Applying a Template to a Page

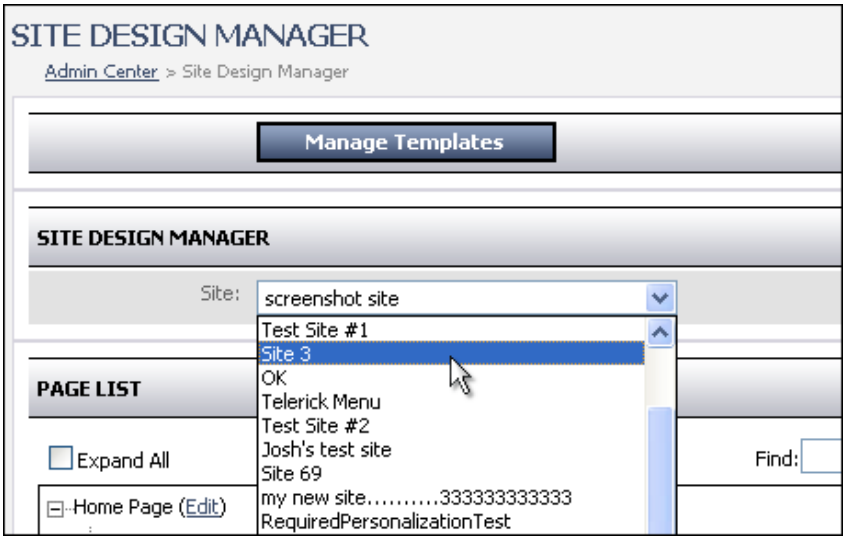
Once a template has been added to a site it can be applied to pages. The Active CM works on an inheritance model so that all new pages, unless otherwise specified, will inherit the templates from the parent page. This inheritance model can be modified at any point and any individual page can be assigned a unique template.

1. Click on Admin Centre in the Toolbar.

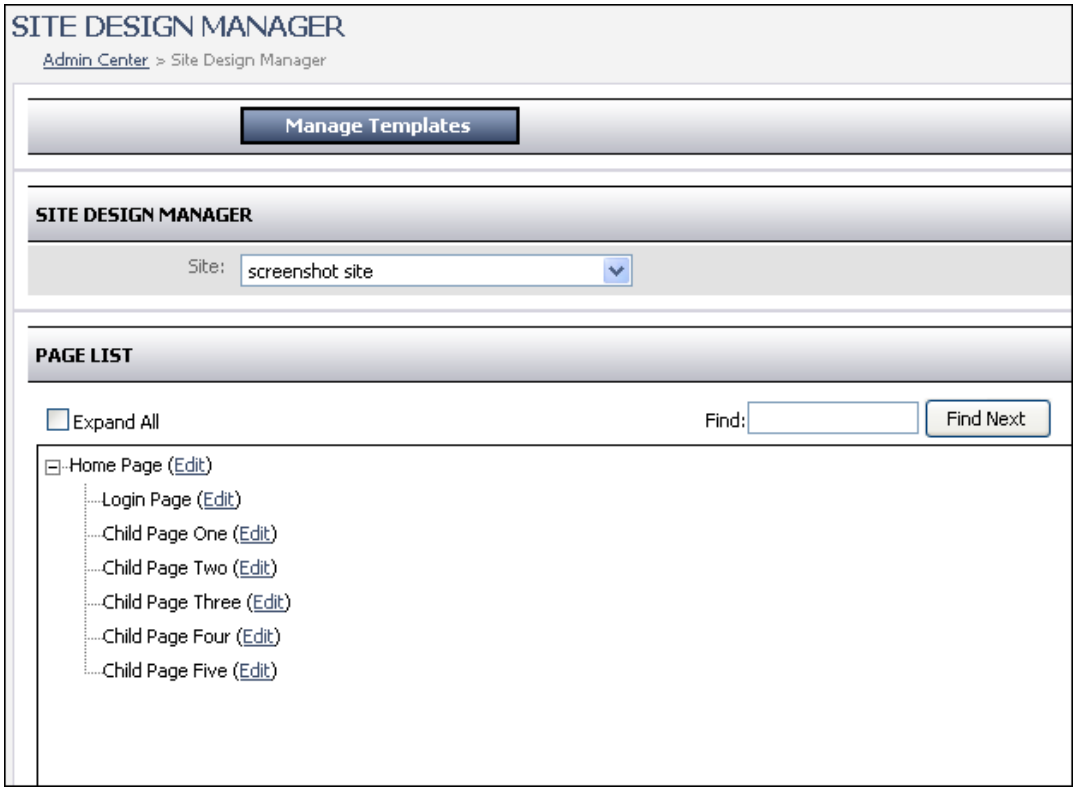
2. Click on the System drop down and click on Site Design



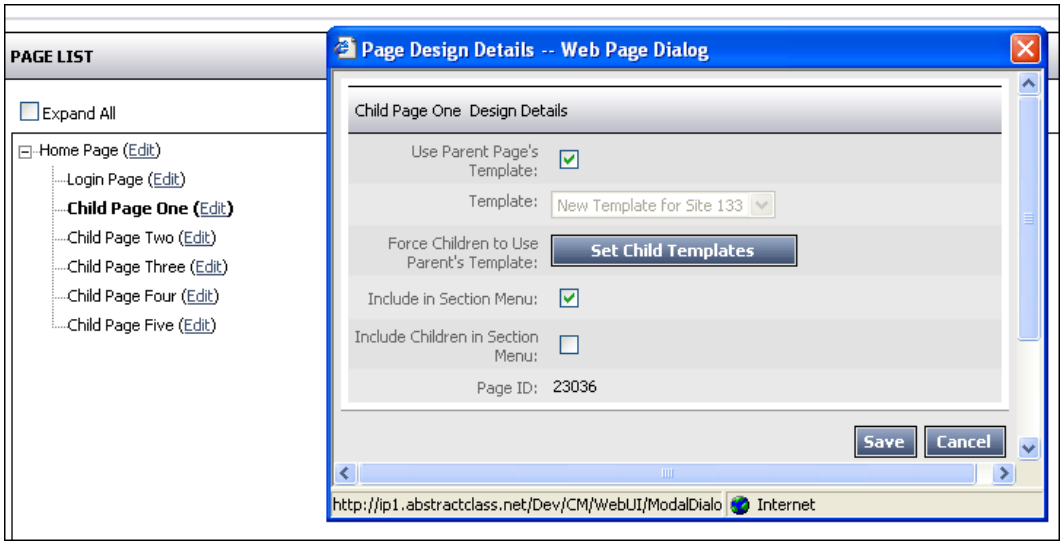
3. If you have multiple sites running on the application, please pick the site you would like to work with.



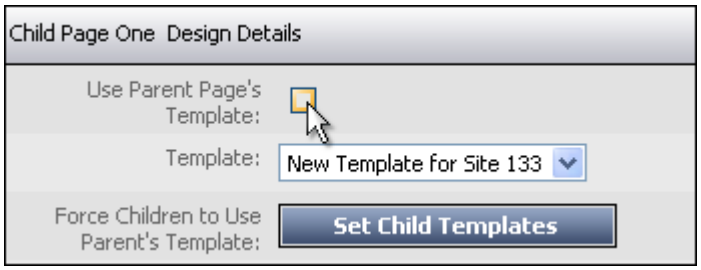
4. In the Page List window, there is a tree view of your entire site.



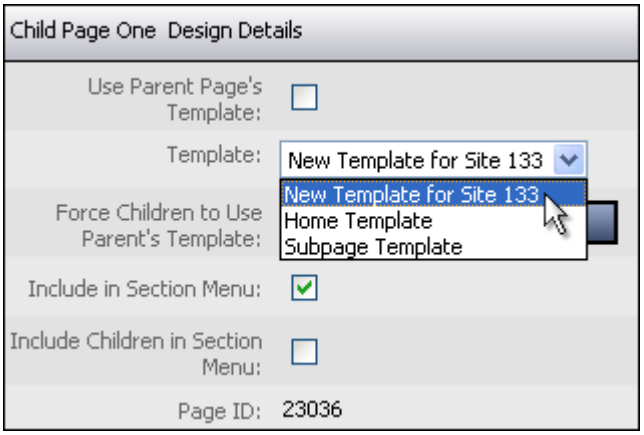
5. Select the page you would like to configure by clicking the edit link beside the page. This will open a 'Page Design Details' window that will allow you to set properties on a specific page, such as the template it will use, and whether or not it will be included in the dynamically generated section menu. By default, the page will have the "Use Parent Page's Template" checked.



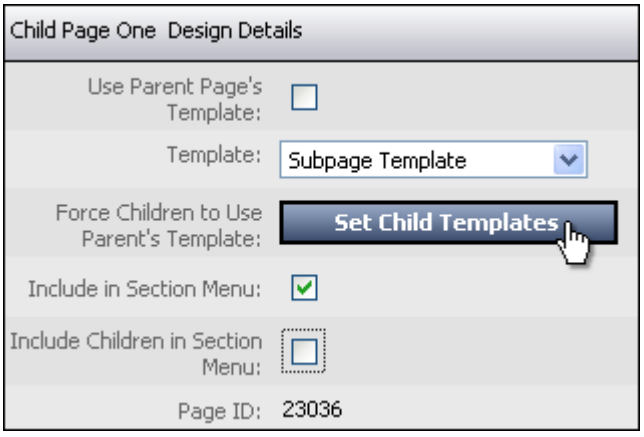
6. To change the template used for this page, first uncheck the “Use Parent Page’s Template” box. This enables the ‘Template’ dropdown menu.



7. Click on the Template drop down and select the template you would like applied to this page



8. You can also choose to force the children of this page to inherit the same template. This makes applying a certain template to an entire section easy.



9. Click save

Reverting to older design templates

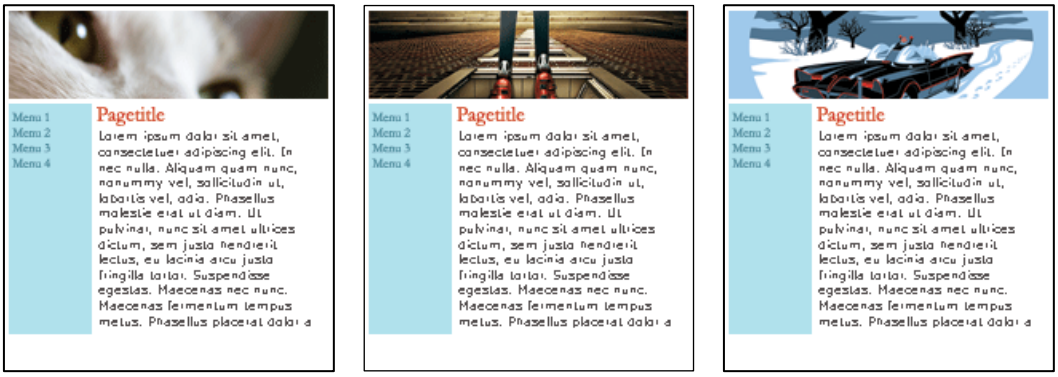
You can revert to any of the older versions of a site's design by first downloading the version of the design package you would like to revert to, and then uploading it again.

Note If you've added templates that are not included in this older version, you will need to update the older SiteDesignManifest.xml to reflect those changes.

Template Variable Association

When creating templates for a site, there are many instances where several templates need to be created to allow for one element to change from section to section. Any changes/maintenance required for the templates have to be made repeatedly on each of the near-identical templates.

For example, each image (below) represents one “section” of the site where unique banner branding is required – in this case, it’s just the banner along the top.



Normally, this would require a brand-new template for each instance. However, the new Template Variable Association allows changes such as images, CSS classes, ID, etc. to be applied to the pages themselves and not have to be hard coded in the templates. Instead, “$VARIABLE$” is placed in the templates where the ‘changing’ element would normally be, and the $VARIABLE$ value is changed depending on what page it’s applied to.

For example, a template design could contain HTML such as:

```
<body class=”$VARIABLE$”>
```

or

```
<img src="images/$VARIABLE$.gif" alt="$VARIABLE$" />
```

If the variable defined for this page is the text "Products", when the ACM renders the page, "$VARIABLE$" would be replaced with "Products" as follows:

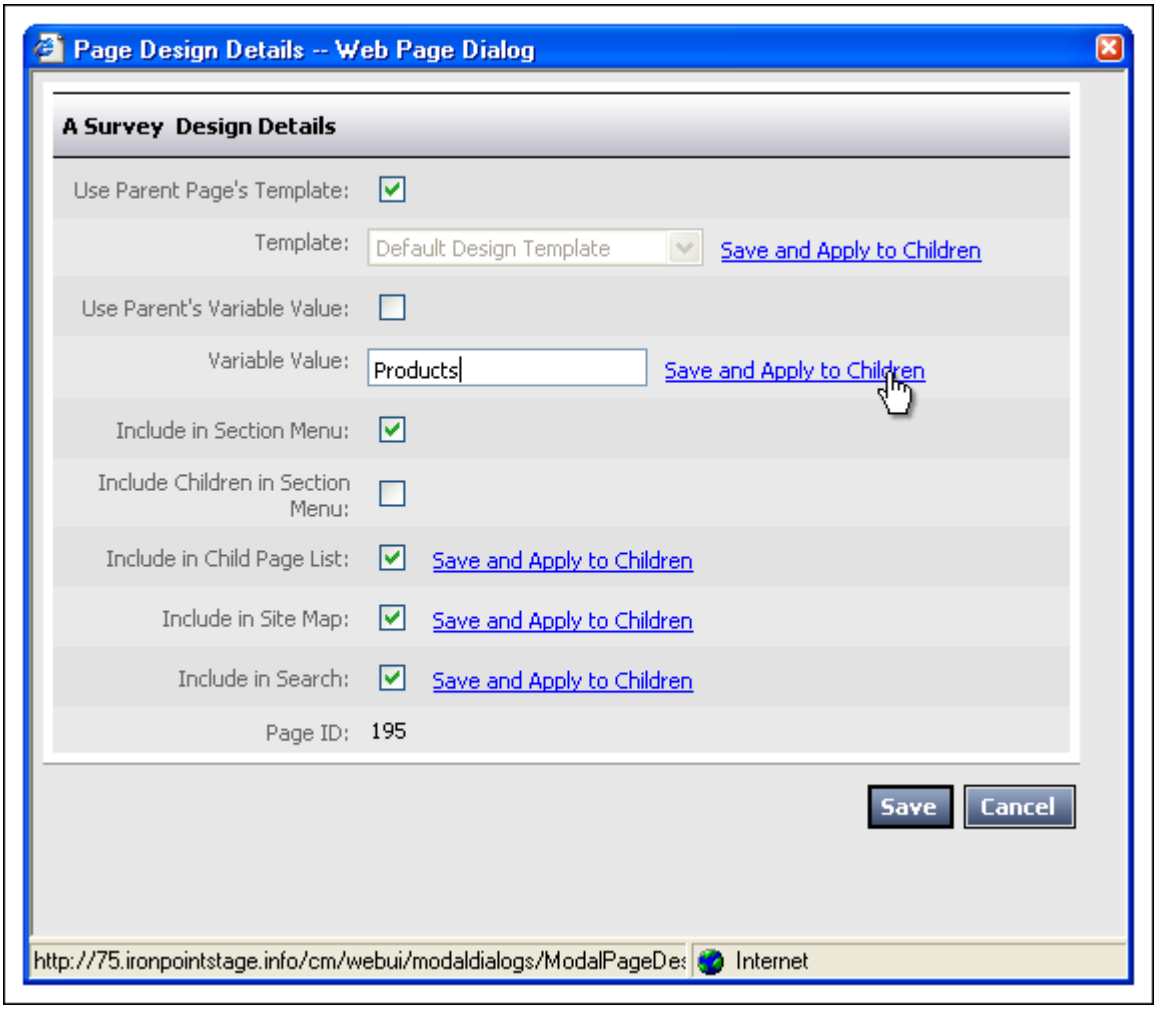
```
<body class="Products">
```

or

```
<img src="images/Products.gif" alt="Products" />
```

The variable for a specific page is set in a new field in the Site Design Manager Page Design Details Dialogue box. The variable value can be entered and forced to inherit

through a page's childpages; when a new page is created, it inherits the parent's template variable value.



The value defined through the Site Design Manager will be used to replace the `"$VARIABLE$"` reference in the associate page's template HTML.

It is important to note that this replacement only occurs in the standard HTML within a template. It does NOT apply to IronPoint tags. For example, `"$VARIABLE$"` in the tags below will not be replaced by the variable value:

```
<ironpoint>$VARIABLE$</ironpoint>
```

```
<ironpoint
RootPageDefID="$VARIABLE$">sectionmenu</ironpoint>
```

Email This Page

In order to configure the Email Page hyperlink, certain steps must be taken. (Note the information provided here is specifically for working with the templates, for complete information on "Email This Page" feature, see the 7.5 release notes).

Template Tag

In order for the hyperlink to appear correctly on a page, it must be added into the template and assigned to the page in the ACM. The hyperlink uses a new "EmailPage" tag, formatted and entered into the associated template HTML file in the standard format, as follows:

```
<IronPoint>EmailPage</IronPoint>
```

The tag has three customizable attributes to allow the designer to specify the text of the hyperlink and the width and height of the popup browser page. These are each specified by the designer within the IronPoint tag, and are named "text", "width", and "height", respectively.

These can be specified as follows:

```
<IronPoint text="Click to Email" width="450"
height="300">EmailPage</IronPoint>
```

Styling is provided for the hyperlink. It makes use of the standard "ipb" class for the hyperlink itself, and is enclosed within a **`<div>`** tag that uses a new "ipf-EmailPage" class, allowing the contents within to be manipulated by designers.

Summary and Reference

Template Tag: `<IronPoint>EmailPage</IronPoint>`

Tag Attributes: `Text, Width, Height`

Example:
```
<IronPoint text="Email" width="450"
height="300">EmailPage</IronPoint>
```

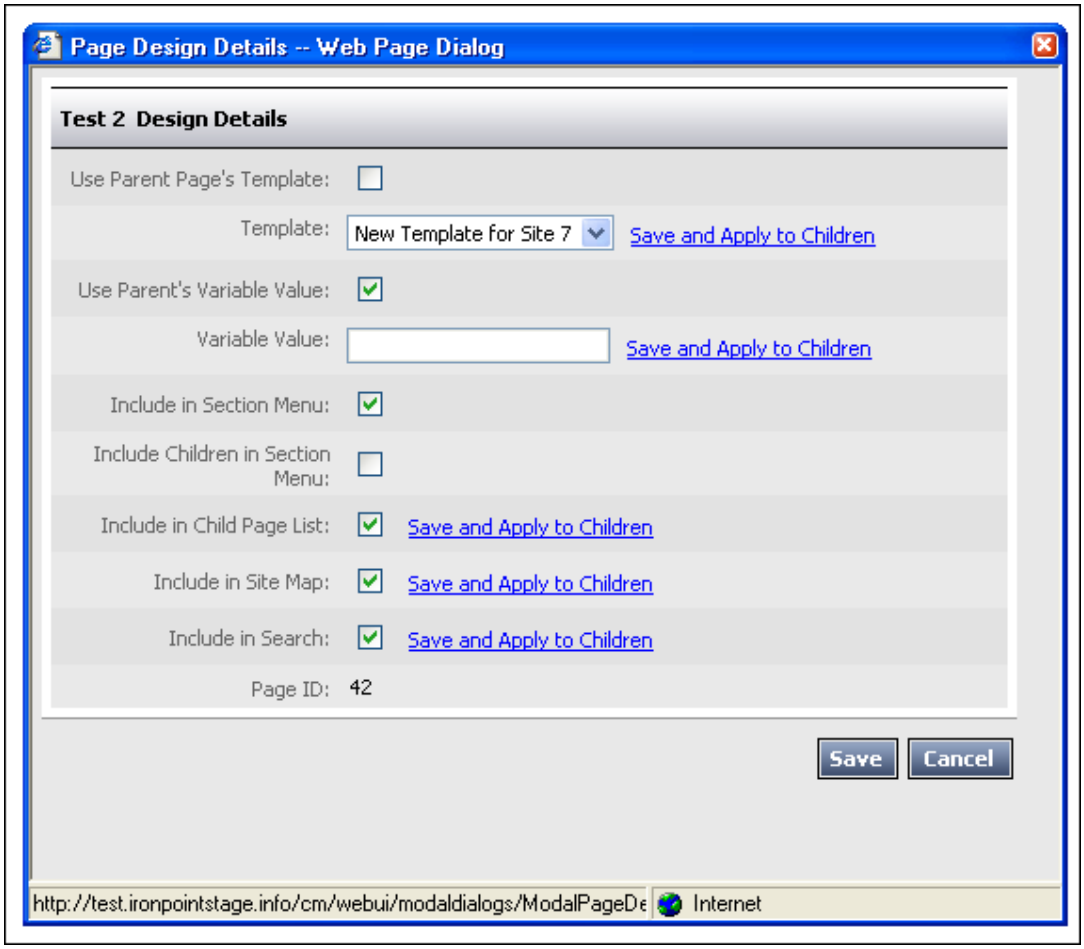
Hyperlink Class: `ipb`

Hyperlink Div Class: `ipf-EmailPage`

Additional Design Details

Three additional controls were added to the Site Design Manager in 7.5.

These tools enable you to hide specific pages and their children from childlink lists, site maps, and the Active CM search tool, and are universal to the site.



Include in Child Page List

This switch allows you to show individual pages and their children in the Child Link lists that are generated with the **`<ironpoint>Childlinks</ironpoint>`** tag in the template, or on the General Content 2 pagetype.

This control was originally available only on a page-by-page basis on the Child Pages tab of Full Edit mode, but was added to the Site Design Manger to centralize the control.

Include in Site Map

This switch allows you to show/hide specific pages and their child pages in any Site Map pagetypes on your site. The enables better control over what specific pages are viewable from the Site Map.

The number of levels that should be displayed in the Site Map is still controlled in the Edit Mode of each individual Site Map page.

Include in Search

This switch allows you to include/hide specific pages and their child pages from the Active CM search tool.

Page Types and the Pagetypes.css

The Active Content Manager uses different 'Page Types' to create different kinds of pages. The most basic Page Type is General Content, which has a single WYSIWYG Editor that allows content providers to enter text, images, links, etc.

Other pages, like the Survey and Calendar pages generate their own HTML (with CSS classes). The default styling for these Page Types is set in the Pagetypes.css in the Design Package. The Pagetypes.css can be edited to style the dynamically generated HTML to match the rest of your website.

Note The **<ironpoint>pagetype</ironpoint>** tag can be used to display the content of most pagetypes. For General Content 2 and General Content 3, two pagetypes that have more than one HTML Editor, the **<ironpoint>pagetype</ironpoint>** will display the content of all editors in one solid block.

If you would like to display the contents of a page's two different editors separately, you can use the **<ironpoint>HTMLContent</ironpoint>** tags.

For specialized pagetypes such as Home 3 (which has three editors), the **<ironpoint>pagetype</ironpoint>** tag won't display any content at all – the **<ironpoint>HTMLContent</ironpoint>** tags must be used.

The **<ironpoint>HTMLContent</ironpoint>** will only display content from General Content and Home pagetypes with more than one editor.

Please see Appendix I: IronPoint Tags for more information.

Advanced Layout Options

To give designers as much flexibility as possible with Page Types that generate HTML, there is an Advanced Layout Option for the pagetypes Quicklinks and Quickpoll.

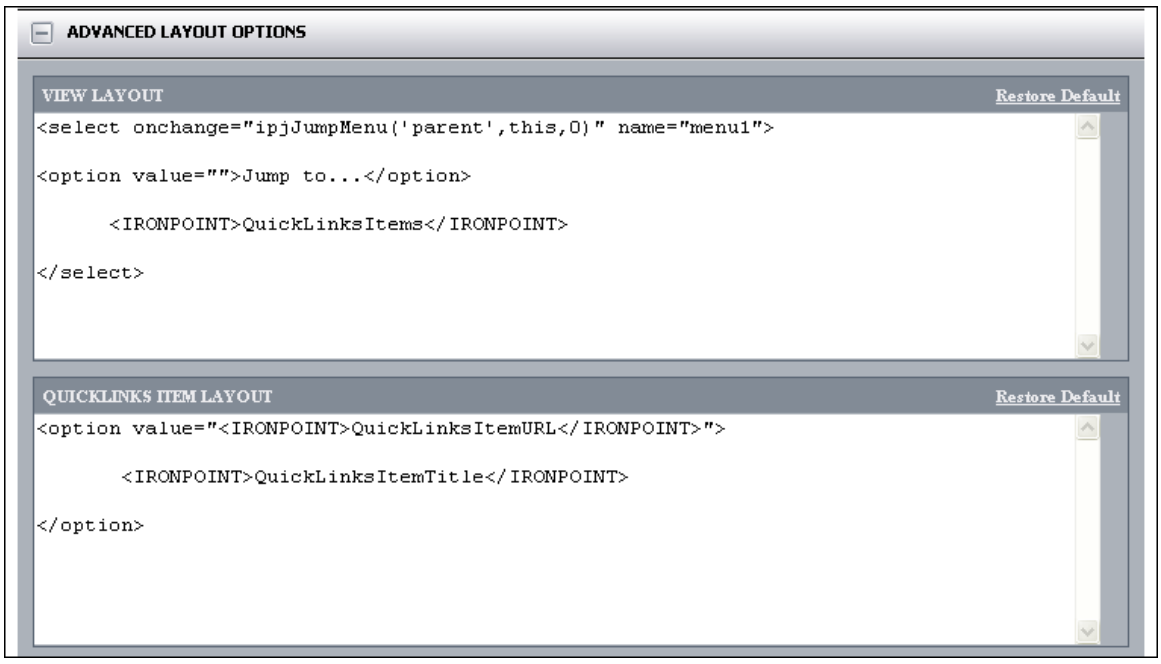
The Advanced Layout Option allows the use of IronPoint tags to display different items of a page type where they're needed. In Edit mode of the Pagetype, the option will be available for expansion at the bottom of the screen.

Each pagetype has specific tags that are used for Advanced Layout – these are available in Appendix II.

The different tags can be used on their own, or combined with HTML to greatly increase design flexibility, as well as options for how a page type will function.

All Advanced Layout options use two or more text fields for layout. With the Quicklinks and Quickpoll page types, these multiple fields allow for the layout of the page type as a whole, as well as the layout of each individual item within the page type.

The example below illustrates how, with the use of the IronPoint tags, HTML and a bit of JavaScript (referencing a function included in the software) a Quicklink Pagetype can be displayed as a 'Jump To…' dropdown menu.



Creating and Adding Pagelets

"Pagelets" are syndicated content used by the Active Content Manager to enable the repeated use of content without having to manage and edit multiple different versions of it. Pagelets are applied to pages by attaching them to 'pagelet zones' on the pagelet tab in Full Edit mode.

To add Pagelets to a page requires a few steps: pagelet zones need to be added to your templates to let you display pagelets. The pagelets themselves need to be created, and then the pagelets have to be applied to pages.

1. Pagelet tags are one of the many IronPoint tags that can be used in your template (for a full list, please see Appendix I). It follows standard IronPoint tag formatting **`<ironpoint>pagelet</ironpoint>.`**

   Every one of the tags added to the template creates a 'zone' to which a pagelet can later be applied. Once a template has been applied to a page, the 'zone' will only be visible in View Mode if there is a pagelet added to it, so you can include several pagelet tags and not have to use them.

2. Creating Pagelets is done in the Syndication Manager, located in the Admin Center. The first screen of the Syndication Manager is a tree view of all the pagelets available on your site, set up much like the tree view of your site in the Site Design Manager. To create a new pagelet:

   a. Click the Edit link next to the Parent 'Pagelets' Item in the tree.

   You will be taken to the Child Pages tab in the Full Edit view of the Pagelets. From this point onwards, it's a lot like creating a new page: select the kind of pagelet you would like from the 'Create New' dropdown.

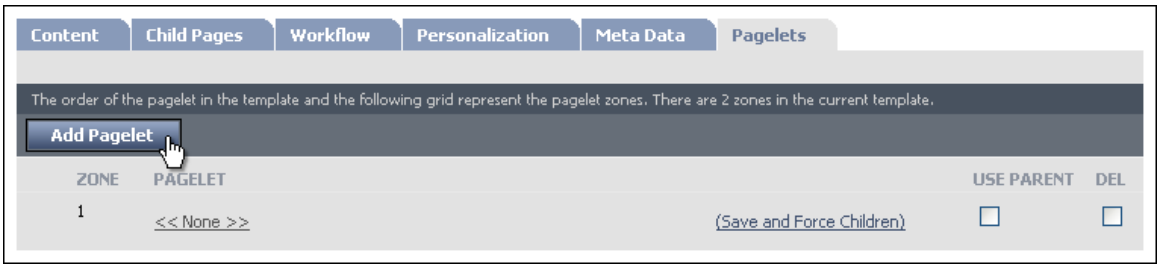
Tip The 'Create New' dropdown displays only the page types that can be created as pagelets. Use the Category page type to organize your pagelets. Pagelets CANNOT be moved once they are created.

   b. On the Sites tab of the new Pagelet, you will be able to limit what sites the pagelet will be available on. The Content tab displays the standard Edit options depending on the kind of pagelet it is; for example, if you've created a General Content pagelet, there will be a WYSIWYG editor with the same functionality of a normal page.

   c. Once you have completed adding content to your pagelet, publish it.

3. **Applying Pagelets** is done in Full Edit Mode of a page.

   a. Click on the Pagelet tab. You can see how many pagelet zones are available in the current template.



b. To 'activate' a Pagelet zone, click 'Add Pagelet'. This puts an empty placeholder in the first available pagelet zone.

c. To add a Pagelet to an activated zone, click on the <<None>> link. This opens a dialogue box that lets you choose which pagelet you would like to add. Once selected, click 'Save'. If you would like to keep the zone empty and add a pagelet to the next one, you can click 'Add Pagelet' again, and leave the first zone at <<None>>.

d. Publish the page.

Tip When the ACM is looking at your template for Pagelet tags, it reads from top to bottom – the first <ironpoint>pagelet</ironpoint> that appears in your HTML becaomes zone 1. This can get kind of confusing when the templates use tables because a pagelet that appears at the top of the design could be the third or fourth zone. When in doubt, look at the HTML.

Template Absolute URLs

By default the ACM generates URL paths as relative. There may be software integration reasons where an absolute path is required. This feature provides the ability to have paths on a page generate as an absolute URL instead of as a relative URL.

Relative URL:

```
<link href="/Sites/3/templates/css/style.css"
rel="stylesheet" type="text/css">
```

Absolute URL:

```
<link
href="http://www.active.com/Sites/3/templates/css/style.c
ss" rel="stylesheet" type="text/css">
```

A comment tag that when inserted into templates acts as a switch to use the external FQDN of the site when creating the absolute path.

1. The tag is **`<!-- IRONPOINT USE ABSOLUTE PATHS -->`** in order to ensure that it is a standard tag and doesn't violate W3C standards for XHTML.
    a. This is a case sensitive tag, it must be entered as ALL CAPS.
    b. Tags entered as lower case will be ignored by the system.
2. The tag should be placed at the very beginning of the template HTML (before the document type declaration).
3. If the tag is added to the template, all links within the final, rendered page will be presented as absolute paths rather than as relative paths. This includes, at minimum, all href, src, and action values.
4. This parsing **does not** include parsing of text within JavaScript variables or within metatags, as it is impossible to identify accurately whether or not inline JavaScript variables and metatag data are intended as relative paths ("Home/Page12.aspx") or not ("text/char").

This feature **will not be supported** on page types which rely on dynamic links, such as survey. While the functionality will not be restricted from these page types, the feature set of the page type relies on dynamic links and is not expected to work properly with this feature enabled. This feature is primarily intended to work for general content and general content-like pages, and will not be available for all page types.

Appendix I

IronPoint Tags

The IronPoint tags are a set of XML tags that are used in the template to place different kinds of content for the site.

All the IronPoint tags are formatted in this way:

```
<ironpoint>TagName</ironpoint>
```

The IronPoint tags include:

PageType

The PageType tag is used to show primary PageType content. This tag works for all templates (front-end, system, report, and modal). The intention is to include one PageType tag per template.

This tag is used to display content of almost all pagetypes except Home 3. When used on pagetypes such as General Content 2 and General Content 3, it will display the content of all editors as one continuous content area.

Attributes:

`ShowChildLinks(String)`

Example: `ShowChildLinks="True"`

Description: If true, the pagetype will show the child links (in the normal location - usually sandwiched between two html editors). Styling is per the default style defined for the ChildLink tag.

Default Value: True

HTMLContent

The HTMLContent tag is used to show editor content from pagetypes with multiple editors, such as the General Content 2, General Content 3 and Home 3. It gives you the ability to use multiple HTMLContent tags per template, and display the content of the editors in different places.

Attributes:

`index(number)`

Example: `index="2"`

Description: Enables you to specify which editor you would like to display content from.

`pageid(number)`

Example: `pageid="12"`

Description: Enables you to specify which page in the system you would like to display the editor content from.

SectionMenu

The SectionMenu tag is used to show the Section Menu. This tag works for front-end pages only. The intention is to support multiple section menu tags per template.

Attributes:

`FileName(string)`

Example: `FileName="SectionMenuData.xml"`

Description: Specifies the xml file to use for configuration of the section menu. Can be set to either SectionMenuData.xml for the ASP.NET 2.0 Menu from ComponentArt, or Panelbar.xml for the Panelbar Menu from Telerik.

Default Value: SectionMenuData.xml

`MenuClass(string)`

Example: `MenuClass="ASPNetMenuFlyOut"`

Description: Specifies the type of section menu to be used. Can be either ASPNetMenuFlyOut or TelerikAccordion.

Default Value: MenuClass="ASPNetMenuFlyOut"

`RootPageDefID(number)`

Example: `RootDefPageID="11"`

Description: Specifies the parent pageDefID. This will allow the children of that parent to show as top level items in the seciton menu.

Default Value: Home Page (usually Page 4)

`RootUrlTitle(string)`

Example: `RootUrlTitle="\home\cityservices\garbage"`

Description: Specifies the parent page (via the UrlTitle) to get the RootPageDefID. See RootPageDefID above.

Default Value: HomePage

`MaxLevels(number)`

Example: `MaxLevels="1"`

Description: Specifies the maximum levels of the section menu to include for this template. Note that this setting overrides anything specified inside the application with regard to including in section menu, etc.

Default Value: 999 (ie. all levels)

LocationLine

The LocationLine tag is used to show the location line. This tag works for front-end pages only. You can insert as many location tags as desired. The LocationLine tag, when rendered, will include links that navigate to the pages shown in the location line.

Attributes:

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

PageTitle

The PageTitle tag is used to show the title of the page. As a default, it also displays the Edit Icons when a user is logged in, although this can be changed through the PageTitle tag's attributes.

Attributes:

`ShowEditIcon(Boolean)`

Example: `ShowEditIcon="False"`

Description: If true, will show the edit icon to the right of the PageTitle.

Default Value: True

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

PageProperty

The PageProperty tag is used to show properties contained in ContentPageBase. This tag only works on front-end pages. You can use as many PageProperty tags as desired.

Attributes:

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

`Property(string)`

Example: `property="CurrentPageVersion.Title"`

Description: Displays specific properties from information from the Active CM application.

"CurrentPageVersion.Title" – Displays the title of the current page.

"IronPointSessionCurrentUser" – Displays the name of the user when they are logged in.

"IronpointSessionCurrentUser.PreferredFullName" – Displays the specified preferred full name of the user when they are logged in.

"CurrentPageVersion.PublishedDateTime" – Displays the date and time from when the current page was last published. (Identifying any time that the page has changed its publication status)

"CurrentPageVersion.ModifiedDateTime" – Displays the date and time from when the current page was last modified. (Identifying any time there's any change to the page)

Default Value: None. You must specify a property when using this tag.

`Format (string)`

Example: `Format="{0}"`

Description: The .NET format string to use for formatting the output. This can be useful for dates. For Example, mmm-ddd-yy

Default Value: {0}

EditIcon

The EditIcon tag displays the Edit Icon. This tag only works for front-end pages. You can insert as many EditIcon tags as desired. If the user clicks on this icon, they will be taken to the edit mode of the page. If the user is not logged in, or does not have rights to edit this page, this icon is not displayed.

Attributes:

`EditText(string)`

Example: `EditText="(click here to edit)"`

Description: Text to use (instead of an image) to go into edit mode. To style, wrap this tag in a div (example:)

Default Value: Empty String – defaults to using icon

`ViewText(string)`

Example: `ViewText="(click here to view)"`

Description: Text to use (instead of an image) to go into view mode. To style, wrap this tag in a div (example:)

Default Value: Empty String – defaults to using icon

`ImageSrc(String)`

Example: `ImageSrc="images/myEditIcon.gif"`

Description: Sets the image to use for the icon.

Default Value: The built in system edit icon (/system/images/icon_edit.gif)

ChildLinks

The ChildLinks tag shows a bulleted list of links representing all child pages of the current page. This tag only works for front-end pages. You can insert as many ChildLinks tags as desired.

Attributes:

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

`ShowSummary(Boolean)`

Example: `ShowSummary="True"`

Description: If true, the summary for the page is displayed.

Default Value: False

`PageDefID(integer)`

Example: `PageDefID="#"`

Description: Specified the PageDefID to pull the child links from.

Default Value: The current page.

SiblingLinks

The SiblingLinks tag shows a bulleted list of links representing all sibling pages of the current page. This tag only works for front-end pages. You can insert as many SiblingLinks tags as desired.

Attributes:

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

ParentChildLinks

The ParentChildLinks tag shows a bulleted list of links representing all children pages of the current page's parent. This tag only works for front-end pages. You can insert as many ParentChildLinks tags as desired.

Pagelet

The Pagelet tag creates a pagelet zone on the page.

Attributes:

`Class(string)`

Example: `Class="MyClass"`

Description: Specifies the css class to use for the div tag that surrounds the location line.

Default Value: No styling.

PrintPage

The PrintPage tag shows a text hyperlink that, when clicked, displays a printer-friendly version of the currently displayed page using the PrintDesignTemplate.htm.

Attributes:

`Text (string)`

Example:
```
Text="Click here for printer-friendly view
of this page."
```

Description: Displays as the hyperlink to be clicked.

Default Value: Print-friendly page

Appendix II

Advanced Layout Tags

Quicklinks Advanced Layout Tags

View Layout:

QuickLinksIntroductionHTML

`<ironpoint>QuickLinksIntroduction</ironpoint>`
Displays the content of the Introduction HTML Editor on the Quicklinks page.

QuickLinksItem

`<ironpoint>QuickLinksItem</ironpoint>`
Displays the layout contents of the QuickLinks Item Layout box repeated for each Quicklink.

QuickLinksConclusionHTML

`<ironpoint>QuickLinksConclusionHTML</ironpoint>`
Displays the content of the Conclusion HTML Editor on the Quicklinks page.

Item Layout:

QuickLinksItemTitle

`<ironpoint>QuickLinksItemTitle</ironpoint>`
Displays the title of the quicklinked page.

QuickLinksItemSummary

`<ironpoint>QuickLinksItemSummary</ironpoint>`
Displays the content of the 'Teaser' field on the Quicklinked page.

QuickLinksItemDate

`<ironpoint>QuickLinksItemDate</ironpoint>`
Displays the date of the Quicklinked page (note: only works for Calendar Event Pagetypes).

QuickLinksItemTarget

`<ironpoint>QuickLinksItemTarget</ironpoint>`
Includes the target property of the Quicklinked item if it has been set on the page (works only for Link Pagetypes).

QuickLinksItemUrl

`<ironpoint>QuickLinksItemUrl</ironpoint>`
Displays the URL to the Quicklinked page.

QuickPoll Advanced Layout Tags

View Layout:

QuickPollQuestionHTML

`<ironpoint>QuickPollQuestionHTML</ironpoint>`
Displays the contents of the Quickpoll Question HTML Editor.

QuickPollItems

<ironpoint>QuickPollItems</ironpoint>
Displays the contents of the Item Layout field.

QuickPollVoteButton

<ironpoint>QuickPollVoteButton</ironpoint>
Displays the Vote button for the Quickpoll.

QuickPollResultsLink

<ironpoint>QuickPollResultsLink</ironpoint>
Displays the link to the results for the Quickpoll.

QuickPollResultsHTML

<ironpoint>QuickPollResultsHTML</ironpoint>
Displays a graph of the Quickpoll results, as rendered in the Current Results field.

Item Layout:

QuickPollItemText

`<ironpoint>QuickPollItemText</ironpoint>`
Displays a radio button, and an item from the QuickPoll Items field.

QuickPollItemGraph

`<ironpoint>QuickPollItemGraph</ironpoint>`

Displays a bar graphic whose width represents the number of votes for each item.

QuickPollItemVotes

`<ironpoint>QuickPollItemVotes</ironpoint>`
Displays the number of votes from an item.

QuickPollItemPercent

`<ironpoint>QuickPollItemPercent</ironpoint>`
Displays the percentage of total votes of an item.

Results Layout:

QuickPollQuestionHTML

`<ironpoint>QuickPollQuestionHTML</ironpoint>`
Displays the contents of the Quickpoll Question HTML Editor.

QuickPollItems

`<ironpoint>QuickPollItems</ironpoint>`
Displays the contents of the Item Layout field.

QuickPollVoteButton

`<ironpoint>QuickPollVoteButton</ironpoint>`
Displays the Vote button for the Quickpoll.

QuickPollResultsLink

`<ironpoint>QuickPollResultsLink</ironpoint>`
Displays the link to the results for the Quickpoll.

QuickPollResultsHTML

`<ironpoint>QuickPollResultsHTML</ironpoint>`
Displays a graph of the Quickpoll results, as rendered in the Current Results field.

Appendix III

SectionMenuData.xml Attributes

Custom <SiteMap> Attributes	
ImagesBaseURL	Folder base for images
Orientation	Vertical or Horizontal
DefaultItemCssClass	Default CSS class applied to each menu item
DefaultItemCssClassOver	Default CSS class applied to each menu item in hover state
DefaultGroupCssClass	Default CSS class applied to entire menu group
DefaultItemTextWrap	Boolean value for text wrap – default is “false”
DefaultItemWidth	Default width for menu item
DefaultCssLevel#	Default CSS class for specific menu level
DefaultCssOverLevel#	Default CSS class for specific menu level in hover state
CascadeCollapse	Whether to collapse the groups in order, starting a parent group's collapse only after its child group has completed its. Values: True (default) False
CollapseDelay	The pause in milliseconds between when a menu loses the focus and collapses. Ie CollapseDelay=”1000”
CollapseDuration	The time in milliseconds that it takes each menu group to fully collapse. Ie CollapseDuration=”1000”
ExpandDelay	The delay in milliseconds before a menu opens. Ie ExpandDelay=”1000”
ExpandDuration	The time in milliseconds that it takes each menu group to fully expand. Ie ExpandDuration=”1000”
ExpandOnClick	Whether to expand the menu only after a click on it and to collapse it only after a click away from of it. Ie ExpandOnClick=”False” (default) ExpandOnClick=”True” (to be used only when parent page is Category pagetype)
DefaultGroupExpandDirection	Direction in which the groups expand. Default is Auto. IE DefaultGroupExpandDirection=”0” (Right) DefaultGroupExpandDirection=”1” (Down) DefaultGroupExpandDirection=”2” (Left) DefaultGroupExpandDirection=”3” (Up)
DefaultGroupExpandOffsetX	Offset along x-axis from groups' normal expand positions. Default is zero. IE DefaultGroupExpandOffsetX=”10”
DefaultGroupExpandOffsetY	Offset along y-axis from groups' normal expand positions. Default is zero.

	IE DefaultGroupExpandOffsetY=”10”
DefaultGroupItemSpacing	Spacing between group items. Default is Unit.Empty. IE DefaultGroupItemSpacing =”10px”
DefaultGroupWidth	Width of groups. Default is Unit.Empty. IE DefaultGroupWidth =”300px” Is over written by DefaultItemWidth if stated first
ShadowColor	Color of the pop-up groups' shadows. Default: CCCCCC IE ShadowColor=”CC6600”
ShadowEnabled	Whether menu's pop-up groups drop shadows. ShadowEnabled=”True” (default) ShadowEnabled=”False” Note: Shadow Value does not render across all browsers
ShadowOffset	Offset of the pop-up groups' shadows. IE ShadowOffset=”20”
TopGroupItemSpacing	Spacing between top group's items. Default is Unit.Empty. IE TopGroupITemSpacing=”10px” Will be overwritten by DefaultGroupItemSpacing if the latter is declared first
<Item> Attributes	
ID=”PageID_#”	Specifies an unique Id for an item, used for selecting the item from program code
Look-ImageUrl *Look-HoverImageUrl* *Look-ActiveImageUrl* *Look-ImageHeight* *Look-ImageWidth*	Image properties that can be set for menu items.
Look-CssClass *Look-HoverCssClass* *Look-ActiveCssClass*	CSS classes that can be specified for menu items.
DefaultSubGroupCssClass	CSS class that can be specified for menu item subgroup.
DefaultSubItemLook-CssClass *DefaultSubItemLook-HoverCssClass* *DefaultSubItemLook-ImageUrl*	CSS classes that can be specified for items below specific menu item.
Look-LeftIconUrl *Look-HoverLeftIconUrl* *Look-ActiveLeftIconUrl* *Look-LeftIconWidth* *Look-LeftIconHeight* *Look-RightIconUrl* *Look-HoverRightIconUrl* *Look-ActiveRightIconUrl* *Look-RightIconWidth* *Look-RightIconHeight*	Icons properties that can be specified for menu items.

Appendix IV

Panelbar.xml Attributes

Custom <Menu> Attributes	
DefaultMenuSpacing	Sets default width between table cells
HeaderCss	CSS class applied to top level menu items
HeaderSelectedCss	CSS class applied to top level menu items in selected state
DefaultItemCss	Default CSS class applied to all menu items
DefaultItemSelectedCss	Default CSS class applied to all menu items in selected state
DefaultExpandEffect	Expand effect used
DefaultToolTip	Tool tip applied to all menu items
DefaultInitialState	Collapsed or Expanded
DefaultCssLevel#	# is a level between 1 and 9
DefaultImageLevel#	# is a level between 1 and 9
DefaultImage2Level#	# is a level between 1 and 9
DefautlImageOverLevel#	# is a level between 1 and 9
TableWidth	Width of accordion menu
<Menu> Attributes	
Validate	**True** - validates the XML content against the schema (default)
	False - does not validate (speeds execution). Turn validation off only when deploying the project.
ImageDirectory	Optional attribute that specifies a base path to the image directory when images are used. (Most useful when the panelbar is embedded in an .ascx page).
<Group> Attributes	
Image	Image to be applied to a whole group of menu items.
ImagePosition	Can be Background, Left, Right, Top, Bottom (see **ImagePosition** above)
Align	Controls the horizontal align of the image. Can be Left, Right, Center)
Valign	Sets the vertical alignment of the Item's text.
<Item> Attributes	
Id	Specifies an unique Id for an item, used for selecting the item from program code
Label	The text that appears on the item.
Href	The URL which will be loaded when the item is selected.
Target	The name of the target window/frame for the Href page to load.
Visible	**True** - makes an item visible (items are visible by default)
	False - hides an item (as of v.2.0 this is applicable to 'header' items as well.
CssClass	Name of CSS class to be applied to the item
ToolTip	Tooltip that will appear when the item is hovered

Image	URL of an image to display on the item.
ImageOver	URL of an image to display when the mouse is over an item (or a 'header')
Image2	URL of an image to display when the item is selected (or when the header item is expanded)
Image2Over	URL of an image to display when the mouse is over a selected item (or over an expanded header)
ImagePosition	Controls how an image is positioned within an item
	Left - Image is displayed in the left corner (beginning) of the item
	Right - Image is displayed in the right corner (end) of the item
	Top - Image is to the top of the Label
	Bottom - Image is to the bottom of the Label
	Background - Image is displayed as a background to the Label
	BeforeLabel - Image is just before the Label (and is subject to the Align attribute) - this is the default image position
	AfterLabel - Image is just after the Label (and is subject to the Align attribute)
State	**Expanded -** displays expanded the item group that belongs to the item.
	Collapsed - displays the group collapsed.
Height	Sets the height (in pixels) of an item
Align	Controls the horizontal align of the Item's Label
	Left - Label is aligned to the left (this is the default setting)
	Right - Label is aligned to the right
	Center - Label is aligned in the center
Valign	Controls the horizontal align of the Item's Label
	Top - The item's content is vertically aligned to the top
	Middle - The item's content is vertically aligned to the middle (default)
	Bottom - The item's content is vertically aligned to the bottom
OnClick	Allows for client side scripting when a user clicks on an item.
OnMouseOver	Allows for client side scripting when the mouse is over an item.
OnMouseOut	Allows for client side scripting when the mouse leaves an item.

Appendix V

Symbols

Here is a chart of the symbols initially available from the 'Insert Symbol' tool in the WYSIWYG Editor. A good resource for these characters can be found at http://www.fileformat.info/info/unicode/char/a.htm.

NAME	SYMBOL	UNICODE
Euro Sign	€	\u20AC
Cent Sign	¢	\u00A2
Pound Sign	£	\u00A3
Yen Sign	¥	\u00A5
Currency Sign	¤	\u00A4
Copyright Sign	©	\u00A9
Registered Sign	®	\u00AE
Trademark Sign	™	\u2122
Plus Minus Sign	±	\u00B1
Not Equal To	≠	\u2260
Almost Equal To	≈	\u2248
Less Than or Equal To	≤	\u2264
Greater Than or Equal To	≥	\u2265
Division Sign	÷	\u00F7
Multiplication Sign	×	\u00D7
Infinity	∞	\u221E
Vulgar Fraction One Half	½	\u00BD
Vulgar Fraction One Quarter	¼	\u00BC
Vulgar Fraction Three Quarters	¾	\u00BE
Superscript Two	²	\u00B2
Superscript Three	³	\u00B3
Per Mille Sign	‰	\u2030
Pilcrow Sign	¶	\u00B6
Section Sign	§	\u00A7
Greek Small Letter Alpha	α	\u03B1
Greek Small Letter Beta	β	\u03B2
Greek Capital Letter Delta	Δ	\u0394
Micro Sign	µ	\u00B5
Greek Capital Letter Omega	Ω	\u03A9
N-Ary Summation	∑	\u2211
Latin Capital O w/Stroke	Ø	\u00D8
Angle	∠	\u2220
Masculine Ordinal Indicator	º	\u00BA
Left-Pointing Double Angle Quotation Mark	«	\u00AB
Right-Pointing Double Angle Quotation Mark	»	\u00BB
Middle Dot	·	\u00B7
Bullet	•	\u2022

Dagger	†	\u2020
Double Dagger	‡	\u2021
Latin Small Letter F With Hook	*ƒ*	\u0192