# How we did it – Building a (sexy) MOSS 2007 WCM

## Rich navigation

by Joe Hardy



Figure : A two-level navigation menu in action on WorleyParsons.com

I won’t lie to you: when the mock-ups came in for WorleyParsons.com, the menus had us panicking just a little bit. The deadline was tight, and the unknowns of attempting to integrate an intricate navigation UI into a Sharepoint site like this were manifold.

We’ll talk about all of this in the article, but here’s an overview of what was used to achieve the job:

* **1x custom menu control** – based on Microsoft’s released SharePoint code (<http://blogs.msdn.com/ecm/archive/2006/12/02/customizing-the-wss-3-0-moss-2007-menu-control-mossmenu-source-code-released.aspx>)
* **1x menu control adapter for our custom menu control** – a largely custom component that was inspired by the brilliant CSSFriendly project (<http://www.codeplex.com/cssfriendly>)
* **2x columns in our base page content type** which stored additional metadata for our menus
* **1x list** used to achieve an optional three-level menu design for some drop-downs
* **1x script** used by administrators to refresh the cache, if necessary
* **Craploads** of CSS!

## Storing extra navigation data

As is illustrated in , the client wanted two columns on every page content type that would contain the additional data required in our navigation:

* **NavImage** stores a basic thumbnail image
* **NavAbstract** is a navigation-specific description



Figure : Highlighting the three components of a navigation item

Every page on the site, regardless of the content type, ultimately inherits from a base content type we called “WorleyParsons Basic Page”. This base content type has NavImage and NavAbstract site columns. These fields can be just dropped into each page layout as required.

## Editorial control

So, the data was easy! Next, we needed to give editors an easy way to add images and abstracts.

We dropped the columns onto each page layout and then had a look at methods for making them visible only to editors.

Our component of choice was Microsoft.SharePoint.Publishing.WebControls.EditModePanel. It does exactly what’s advertised: displays content when the user is in edit mode.

<PublishingWebControls:editmodepanel PageDisplayMode="Edit" runat="server">
 <SharePointWebControls:textfield InputFieldLabel="Navigation Item Abstract" FieldName="NavAbstract" runat="server"></SharePointWebControls:TextField>
 <PublishingWebControls:richimagefield InputFieldLabel="Navigation Item Image" FieldName="NavImage" runat="server"></PublishingWebControls:RichImageField>
</PublishingWebControls:EditModePanel>



Figure : Editing the navigation content

## A curve ball – optional three level navigation menus

The first real challenge was the client’s requirement that one drop-down menu in particular needed to have a design that would accommodate three levels of content. This would allow users to navigate straight to specific subpages. The menu is illustrated in .



Figure : Three level navigation

We decided that we wanted site administrators to easily configure the style of menu for any site in the top navigation. How should we store the settings?

In the end we elected to use a SharePoint list that stored URLs of top-level navigation items that would display the alternate menu design. We called the list **ThreePartNavUrls** and referred to it by name in our MenuAdapter code.



Figure : The list, containing a URL reference to our top-level page for the three-level menu

We simply store the URL in the Title field. Nothing more was required.

## Introducing control adapters

If you drop a SharePoint AspMenu control onto a page and view the source it generates, you’ll quickly realise two things:

1. The generated HTML can easily account for over 50kB of bloated code – without counting the associated scripting and CSS files – thanks to its inefficient use of tables and inline styling
2. The code is also virtually impossible to style cleanly, and is far from lightweight when load on the browser’s rendering engine is considered.

So what to do? We wanted a menu that was relatively standards compliant, lighter in weight and still had the advanced styling we wanted to achieve. It was becoming clear that AspMenu out of the box wouldn’t give us what we needed – and nor would many of the third party menu controls we evaluated.

Enter ASP.NET control adapters.

Scott Guthrie’s blog has a good starting point for learning about the control adapter architecture at <http://weblogs.asp.net/scottgu/archive/2005/12/21/asp-net-2-0-control-adapter-architecture.aspx>. He also points to the CSSFriendly project which served as a building block for a number of the adapters we ended up creating.

Control adapters don’t seem to get as much attention as they deserve, but they offer a world of flexibility when it comes to customizing output from ASP.NET web controls. In this case they allowed us to override the standard ASP.NET Menu Render() methods with our own, which provided cleanly formatted output. They can be dropped into web applications and removed without changing a line of code, making them a simple, modular approach.

Control adapters integrate tightly with .BROWSER files which means that you can cater for different browsing platforms by simply dropping additional adapters into your environment.

If you’re interested in reading up further on how .BROWSER files work, check out “Browser Definition File Schema” at <http://msdn.microsoft.com/en-us/library/ms228122.aspx>.

The following components were developed to make this work:

|  |  |
| --- | --- |
| Class Name | Description |
| WorleyParsons.Controls.WpMenu | This is a modified version of the source code Microsoft released for Microsoft.SharePoint.AspMenu.  |
| WorleyParsons.Adapters.MenuAdapter | Originally based on CssFriendly.MenuAdapter. All of our custom rendering logic is contained within this class. We ended up bundling all querying of the SharePoint object model into this adapter. |

Table : Components used to make the menu adapters happen

After we developed the components, we simply dropped a reference into a .BROWSER file, which then made the control’s rendering hierarchy bypass the Menu control’s internal Render() functions and use ours instead.

Here’s an excerpt from our WorleyParsons.Adapters.browser:

<browsers>

 <browser refID="Default">

 <controlAdapters>

 <adapter controlType="WorleyParsons.Adapters.WpMenu" adapterType="WorleyParsons.Adapters.MenuAdapter" />

 </controlAdapters>

 </browser>

</browsers>

## How to use the SharePoint object model to query site data

Time was against us with this project, which meant that we weren’t able to do anything tricky like create support for custom menu item templates or conditional formatting directives, to make it easier to change the menu’s look and feel down the track. As a result, we had to go for the less desirable option of embedding some of the layout information and data awareness in the actual control adapter.

As well as this, querying of the SharePoint database happened directly within the control adapter. We’d probably recommend an abstraction layer that is better future-proofed for changes down the track, but from a performance point of view we ended up being pretty happy with how things worked out.

But not without pain!

**Newbie note:** For those new to the SharePoint object model, two of the main objects you use to access information in SharePoint are SPWeb and SPSite. To query information pertaining to the user’s current session, you can access these objects via SPContext.Current, which is similar to HttpContext.Current in ASP.NET.

Handling memory leaks caused by use of the SPSite and SPWeb classes ended up being a nightmare that caused a few sleepless nights. The following articles proved invaluable in helping us out of some common pitfalls that made our code to inexplicably bring servers to their knees or mysteriously send application pools into a stupor.

* <http://blogs.technet.com/stefan_gossner/archive/2008/05/07/troubleshooting-spsite-spweb-leaks-in-wss-v3-and-moss-2007.aspx> - Author: Stefan Gossner
* <http://www.sharepointnutsandbolts.com/2008/06/disposing-sharepoint-objects-what-they.html> - Author: Chris O’Brien

**Debugging memory leaks in the SharePoint object is a difficult process.** You will frequently find that code crashes without much explanation, and your best hopes of resolving issues without too many problems is to ensure that you enable logging and stack traces (as Stefan Gossner describes in his article above) and have a go at line-by-line debugging on your SharePoint server.

The worst pitfall we encountered was when using the SPWeb.Webs collection: our application would inexplicably crash at (seemingly) random times and wouldn’t give us any additional information beyond a simple “401 Access Denied” error (consulting the error log confused matters further by claiming that we had too many SPRequest objects open)

It was never mentioned on any of the blogs, but we ended up finding that the problem was caused by the component running on behalf of users with limited authentication privileges. When attempting to enumerate all of the webs on the site, a web with restricted access would cause the component to crash with the aforementioned 401 error.

The solution was to use SPWeb.GetSubwebsForCurrentUser() instead of the general SPWeb.Webs collection. Once this was implemented our component ran without any hiccups!

## The importance of caching

SharePoint is *extremely* touchy about how many SPSite and SPWeb objects are being held in memory, so even when you Close and Dispose your objects religiously you are bound to encounter problems once your site starts getting hit by a steady stream of concurrent users.

**Make sure you cache any data that you pull from these objects.** We ended up caching *everything* in the menu for 2 hours.

We then provided a script which we installed to the /\_layouts directory that allows administrators to refresh the navigation cache at any time. We provided a shortcut in their Site Actions menu (as illustrates.)



Figure : A shortcut for administrators to refresh the navigation cache