24 DAYS IN UMBRACO

2012 edition

Foreword

This is a collection of 24 articles that comprised the Christmas Calendar of 2012, for and by the Umbraco Community, called “24 Days In Umbraco”.

It was great fun and there were lots of tips & tricks for every one, so we thought it would be nice to be able to take it with you, thus creating this eBook directly from the source.

Credits

We'd like to thank **Anders Stentebjerg** for designing the cover in the nick of time. A very special thank you goes to **Niels Hartvig** for being so cool, and of course every single one of the authors!

— 'til next year,
Jan Skovgaard, Sebastian Dammark & Chriztian Steinmeier

Table of Contents

The Tale of the 3 CMSs

— by Claus Hingebjerg

Once upon a time there was a small danish agency. It was the late 1990s and the agency, as most other agencies, was the proud owner of its own CMS.

Over the years the CMS grew and grew. The clients were happy, but over time the developers grew tired of it, and one day they were blessed with a new thing: *.NET*. A new CMS was born. And again the clients were happy, and the devs were happy.

Years went by and everybody were happy. Well, almost everybody. The front end guy got more and more annoyed with the CMS. He felt locked-in and dependant on the devs. Unable to do what he wanted, he demanded a new CMS. The devs agreed with him, mostly because they were tired of supporting him in the smallest of things and sick of his whining: *"Why can't I do this?", "Why can't I have that?"*.

So, after some time, the devs decided to write a new version of the CMS. But the front end guy said: *"Why write our own, when there are so many established CMSs on the market?"*. *"Fine!"*, the devs said, *"but it has to be .NET"*. The team tested the most popular CMSs on the market, but were not impressed. The APIs didn't impress the devs. The template engines and editor functions didn't impress the front end guy. The licenses didn't impress the bosses and the clients.

Back at the drawing board, one of the devs mailed a friend at Microsoft to get some pointers for the new version of the agency's CMS. The Microsoft guy pulled out his magic wand, whacked the dev on the head and said: *"STOP! Do not create another version of your CMS! Use Umbraco! It's .NET! It's Open Source! It's Danish!"*

And so it came to be, in the year 2008, that the agency started using Umbraco 3 as their one and only CMS. The devs were happy. They were blessed with a solid base to build upon. The front end guy was happy. He could do all the things he ever wished for with his markup, and he could build the backend to suit his clients. The client were happy. They were treated to a backend tailor-made to suit their needs.

4 years and about 100 sites later, both the agency and Umbraco are still going strong, building sites to great satisfaction for both themselves and their clients.

And if all goes well, they will live happily ever after.

Merry Christmas,

Claus Hingebjerg, Front end guy.

Using DocTypeMixins

— by David Conlisk

Hi all!

Here is a short screencast on one of my favourite packages for Umbraco, DocTypeMixins. It's just one great example of the packages created by members of the Umbraco community that make using Umbraco such a joy. Apologies to anyone who saw me give this presentation at the last Glasgow Umbraco Users Group (GLUUG)!

Online Video Clip

The package was written by Pete Duncanson and Matt Brailsford, and you can get it here on our.umbraco.org.

Happy Christmas!

David

1-1 Multilingual Websites in Umbraco

— by Mads Jørgensen

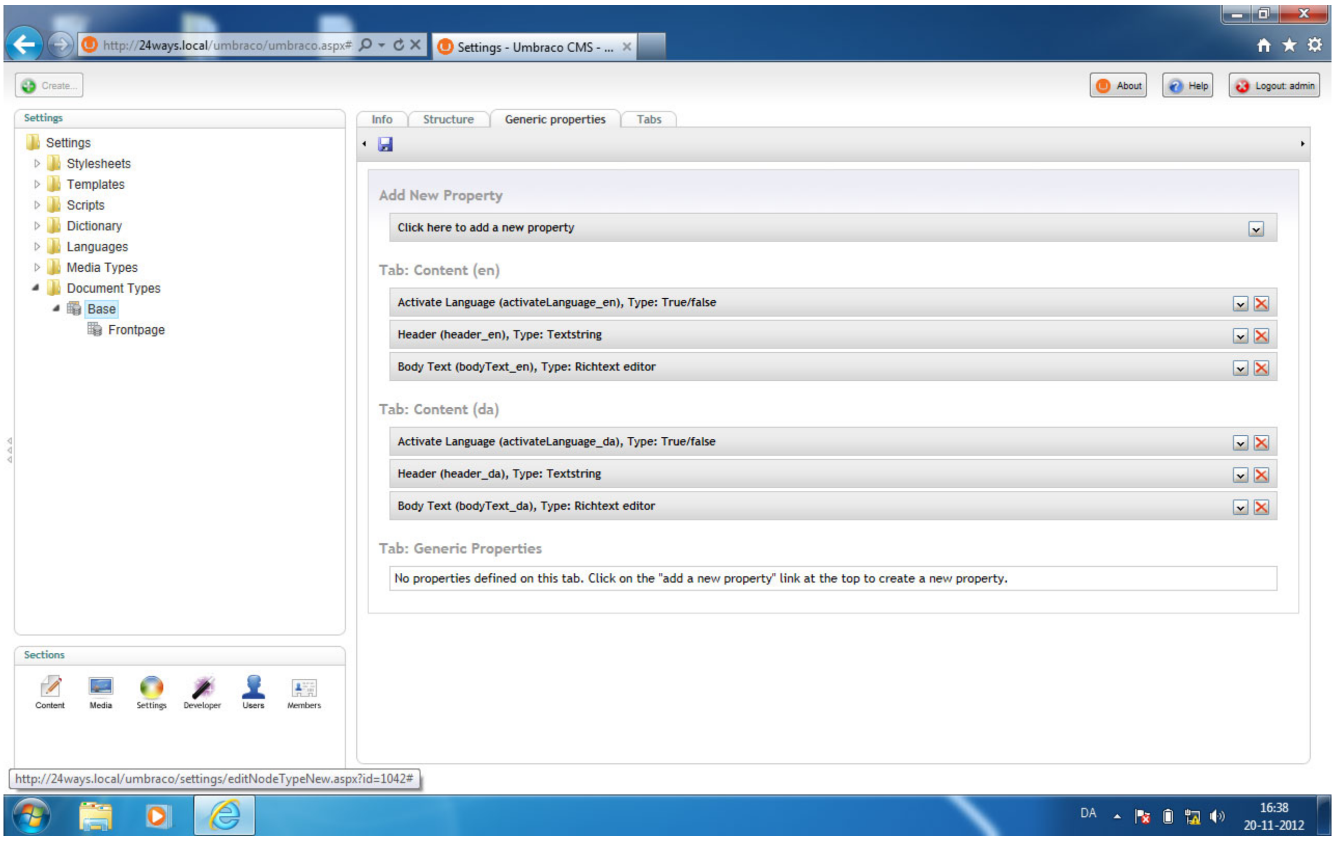
Classic challenge in Umbraco: You have a 1-1 multilingual website, but Umbraco isn't quite made for handling that. This is how we, at ILLUMI, normally handle that. Usually there's quite a bit of custom stuff to handle, as they're not quite 1-1 anyways, but that I won't cover in this article.

What are 1-1 multilingual websites?

Quite simply, it's when you have the exact same pages in every language. A CMS like Sitecore would handle it with the ability to version any given item in several languages by clicking a flag in the backend, and then use culturecodes to fetch the right language.

How to handle it in Umbraco's backoffice

First off, we're giving the user several tabs in any document template representing the different languages. Then what we do, is to postfix all our fields with the culture code when creating them in the document type. See screenshot for my example.



Add New Property

Click here to add a new property

Tab: Content (en)

Activate Language (activateLanguage_en), Type: True/false

Header (header_en), Type: Textstring

Body Text (bodyText_en), Type: Richtext editor

Tab: Content (da)

Activate Language (activateLanguage_da), Type: True/false

Header (header_da), Type: Textstring

Body Text (bodyText_da), Type: Richtext editor

Tab: Generic Properties

No properties defined on this tab. Click on the "add a new property" link at the top to create a new property.

Properties setup for multiple languages

When postfixing, we have the opportunity to fetch them from our macros (I'll be doing it in XSLT).

If every page doesn't have all languages, you could make a boolean field, that "activates" the language edition. This will typically help you in your navigation macros, so the pages unavailable in the current language actually won't show in navigation.

Detecting current language from XSLT

So, for some odd reason there's no extension method for fetching the current culture in default Umbraco (I know there are packages around doing that). But there's a very simple hack you could do, and I quite like it.
You simply create a dictionary item, I call it **currentCulture**, and then fill your field postfix texts in each language (**da** and **en**).

Helper XSLTs

I really like to make and use default helper XSLTs (thanks a bunch @greystate for the inspiration). This is basically a seperate XSLT file I include in pages where needed, containing some templates and some variables I use sitewise. In this case I use

this: _MultiLingualHelper.xslt. The methods I'll be storing in this file are:
- Language detection (variable named `$lang`)
- Complete postfix for fields (`$lang_postfix`)
- Fallback language
- Some default settings to generate some content.

Fetching the right version of the field

Now for fetching the right version of a field, I'll use some XPath.

```
<xsl:value-of select="*[name() = concat('header', $lang_postfix)]" />
```

Getting the language-specific version of a property

Explained shortly, the concat part adds **_da** or **_en** to the fieldname, and we then fetch the field that has that name. Easy peasy. See NavigationExample.xslt in line 24 for further inspiration.

Setting up hostnames

Now for the system to detect which language/culture the user wants, we use sub-domains, e.g.: **www**.domain.com and **da**.domain.com. Simply use native umbraco features to set up *hostheaders* per domain.

Is the sub-domain or extension domain necessary? Yeah, I do believe it is, since you have to be very careful not to make duplicate content, as Google won't like that. Of course there is the opportunity to do it all without domain changes, but I like the clear and obvious solution it provides.

Navigation

Navigations in a multilingual setup is basically not that different from default navigation snippets. The main difference is in this line:

```
<xsl:apply-templates
    select="$currentPage/
        ancestor-or-self::Frontpage/
        *[*[name() = concat('activateLanguage', $lang_postfix)] = 1]"
    mode="page"
/>
```

Selecting activated pages for a language in Navigations

It's from NavigationExample.xslt line 16. What it basically does, is use XPath to fetch only pages that are activated through the checkbox.

Changing language

This is one of the more difficult things. What we need to consider is, if the actual page I'm viewing is available in another wanted language, link to the same page, but with the other language's domain. Otherwise link to the other language's frontpage.

See _MultiLingualHelper.xslt for templates doing this.

That's how I do 1-1 multilingual websites in Umbraco. I believe there could be more to it, if you have loads of special features. I recommend looking into differencing date- and currencyformats per language as well, but it won't be covered here.

Any questions, ideas and comments, please contact me via my twitter handle @madjor5. If you want to see what else I'm up to, take a look at illumi.dk for further information.

How to view source with pride

— by Dan Okkels Brendstrup

Or: **How I learned to accept Master Pages, but leave them as quickly as possible**

Content management systems are good for many things, but outputting nicely indented markup is rarely one of them.

You know the feeling, don't you? You spend all your time crafting templates full of lean, semantic, future-friendly markup, only to view source on the final site and discover HTML that appears to have been thrown together by chimps with shovels.

This happens even with Umbraco, which, unless you do something wrong, gives you complete control over your markup. Alas, this simply appears to be the nature of template-driven content management.

But as proud craftsmen of the web we shouldn't settle for second-rate output from a first-class tool, so let's set out to right this wrong.

Empty lines? We don't need no stinkin' empty lines

Do a quick view-source on your latest Umbraco project. I bet you'll find that the page starts with a blank line before the doctype, right? And ends with another blank line at the end of the document. Important? Hardly (though a dangerous step towards triggering quirksmode in IE). Pretty? Not really.

It's easy to fix, though. A linebreak after the opening or before the closing `<asp:Content>` tag will output a blank line in your final markup, so you simply need to concatenate the lines in your Master Page file like this:

```
<%@ Master Language="C#" MasterPageFile="~/umbraco/masterpages/default.master"
AutoEventWireup="true" %>
<asp:Content ContentPlaceHolderID="ContentPlaceHolderDefault" runat="server"><!DOCTYPE html>
[insert greatness here]
</html></asp:Content>
```

How to avoid blank lines in Master Pages

Yes, you're sacrificing proper indentation of your Master Page file, but you're gaining proper formatting of your site's HTML and the respect of all your source-viewing nerd friends. Worth it, right?

Flush left or flushed down the drain

So let's get out of this Master Page and into some macros, shall we? Easy:

```
<html>
  <head>
    <umbraco:Macro Alias="Head" runat="server"/>
  </head>
  <body>
    <div class="wrapper">
      <umbraco:Macro Alias="Header" runat="server"/>
      <div id="content" role="main">
        <asp:ContentPlaceHolder Id="mainContent" runat="server" />
    </div>
  </body>
</html>
```

A simple template with macro tags

Yeah, you wish it were that easy, don't you. View source again, and this is what you get:

```
<html>
  <head>
    <title>A content page | Best Website Ever</title>
<meta name="description" content="" />
<meta name="keywords" content="" />
  </head>
  <body>
    <div class="wrapper">
      <header role="banner">
<a href="/" title="Home" accesskey="1">Best Website Ever</a>
</header>
      <div id="content" role="main">
        <h1>Here is my content</h1>
<p>But oh wait, why is it suddenly indented all wrong?</p>
<p>Argh, I fail at teh internets!</p>
      </div>
    </div>
  </body>
</html>
```

A simple template... with borked indenting

Oh the pain. The first line of your macro is indented perfectly, but your Master Page will be damned if it cares about the rest of your content.

The solution is to keep all tags in your Master Page flush left. Since this is at odds with truly perfect indenting of your markup, you'll do best to have as little markup in your

Master Page as possible and keep the rest in your macros:

```
<html>
<head>
<umbraco:Macro Alias="Head" runat="server"/>
</head>
<body>
<div class="wrapper">
<umbraco:Macro Alias="Header" runat="server"/>
<div id="content" role="main">
<asp:ContentPlaceHolder Id="mainContent" runat="server" />
</div>
</body>
</html>
```

Keep all tags flush left in your Master Pages

Remember that all nested ContentPlaceHolder templates must also concatenate lines to avoid blank lines in the output:

```
<asp:Content ContentPlaceHolderId="mainContent" runat="server"><umbraco:Macro Alias="SubNavigation"
runat="server"/>
<umbraco:Macro Alias="Article" runat="server"/></asp:Content>
```

Remember to avoid blank lines in nested Master Pages as well

Indent? Yes. How can you even ask?

After having whipped our Master Page into shape, let's not let XSLT ruin the party. Thankfully, XSLT was built to create nicely formatted SGML output, as long as we remember to ask politely. You've probably seen this line at the top of all XSLT files that you create via Umbraco's backend:

```
<xsl:output method="xml" omit-xml-declaration="yes"/>
```

An XSLT output tag without explicit indenting instructions

This instructs the XSLT processor to output well-formed XML, but it is missing a crucial attribute. Change it to this:

```
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes"/>
```

A simple attribute gets you nicely indented XSLT output

...and you've told the XSLT processor that it may add additional whitespace to your output, which in most cases will mean that the processor will do what it can to nicely indent your output.

If we apply these tricks to our page from before, we'll end up with something more pleasing to the eye:

```
<html>
<head>
  <title>A content page | Best Website Ever</title>
  <meta name="description" content="" />
  <meta name="keywords" content="" />
</head>
<body>
<div class="wrapper">
  <header role="banner">
    <a href="/" title="Home" accesskey="1">Best Website Ever</a>
  </header>
  <div id="content" role="main">
    <h1>Here is my content</h1>
    <p>And look, my markup is now nicely indented.</p>
    <p>Gee whiz, I might just win at teh internets after all!</p>
  </div>
</div>
</body>
</html>
```

Behold the final, nicely indented HTML output

So go forth and indent, and take even more pride in the fruits of your labor.

And if you have any tricks of your own for fine-tuning markup, we'd love to hear them in the comments!

Introducing The umbraco:image Control

— by Morten Bock Sørensen

As we all know, Umbraco is the friendly CMS. And maybe it is because of our fantastic community, that the act of outputting an image on a page has almost always caused a google search, and a bit of head scratching. Maybe followed by a forum post, and a pointer from one of the many helpful community members.

The scenario is: You have a documenttype that contains a media picker. Now you would like to output the image that was picked to your page. The first time you try that, you probably did this:

```
<umbraco:item runat="server" field="bannerImage" />
```

Using the standard umbraco:item control

And you probably got a bit disappointed when it returned "1063", and not an image. Then you go to google and figure out that you need a macro of some sorts, or some inline XSLT mumbo-jumbo, and after a while, you kind of just accept that.

No more!

As of Umbraco 4.11, there is a new control in town: **umbraco:image**. It is fairly simple to use, and should make it a bit easier to work with images. You use it like this:

```
<umbraco:Image runat="server" field="bannerImage" width="200" height="100" class="banner" />
```

Using the new umbraco:image control

And that will output this:

```
<img src="/media/19/73006.jpg" width="200" height="100" class="banner" />
```

Default output with no parameters

ZOMG! That is the most awesome thing since sliced bread. Almost :-) Let's take it a little further.

There are three attributes on the control:

- field - The property to get the image id or path from
- parameters - A name/value collection passed to the selected provider. (Default provider supports thumb|crop)
- provider - Specifies which provider to use when generating the url for the image

All other attributes will be passed directly through to the img tag. For example the width and height attributes.

Providers

Per default, the control will use the built-in provider, which will allow you to select a thumb that is generated by the Upload datatype or a crop from the ImageCropper to display. You would do that like this:

```
<umbraco:Image runat="server" field="bannerImage" Parameters="thumb=200" />
```

Built in thumbnail usage

Which generates

```
<img src="/media/19/73006_thumb_200.jpg">
```

Rendering with thumbnail specified

or

```
<umbraco:Image runat="server" field="bannerImage" Parameters="crop=small" />
```

Using the small crop from the cropping datatype

Which generates

```
<img src="/media/19/73006_small.jpg">
```

Rendering with a crop specified

Roll your own

Many of you are using other forms of image tools, such as the lovely ImageGen. No problem! You can write a provider that will generate any type of URL you want! I have written a quick sample provider for ImageGen here, that you could take a bit of inspiration from: https://gist.github.com/3835255

To use a specific provider, just tell the control to do so:

```
<umbraco:Image runat="server" field="bannerImage" Parameters="width=200" Provider="imageGen" />
```

Specify custom provider

And with my custom provider, that would give you this output:

```
<img src="/ImageGen.ashx?image=/media/19/73006.jpg&amp;width=200">
```

Output from custom ImageGen provider

And Bob's you uncle! :-)

I hope you will find this useful, and maybe give some feedback on how you like it.

Ajax Paging Using AltTemplates

— by Matt Brailsford

I'm sure we all know the joys of paging content, usually entailing reloading the page and passing a "page" querystring parameter through to say which page of content to load next. It works, but it's just not as sexy as an ajax one, but then you are talking about a whole heap load more work. Well, not with Umbraco and the use of altTemplates.

Step 1: Setup regular old paging

So to start with, just create the normal querystring based paged content. This will be our fallback, so if people don't have javascript enabled, our paged content will still work. Did I mention it was unobtrusive? no? oh, well, it is :)

I won't go in to detail on how to do this as there are plenty of examples, but all I will recommend is to encapsulate your paged content within an Umbraco Macro as we are going to reuse it again in a moment.

Step 2: Wrap your paged content area

As we are going to want to get a handle on the paged content block via javascript, wrap your paged content Macro with a html element and set its ID to something memorable.

You should now have a main template looking something along these lines:

```
<%@ Master Language="C#" MasterPageFile="~/umbraco/masterpages/default.master"
AutoEventWireup="true" %>
<asp:Content ContentPlaceHolderID="ContentPlaceHolderDefault" runat="server">
<html>
    <head>...</head>
    <body>
    ...
        <div id="page-content">
            <umbraco:Macro Alias="PagedContent" runat="server">
        </div>
    ...
    </body>
</html>
```

```
</asp:Content>
```

MainTemplate.master

Step 3: Create an altTemplate

Next up, we are going to make an altTemplate. This template needs to be really clean and should simply contain your paged content macro. THAT'S IT.

```
<%@ Master Language="C#" MasterPageFile="~/umbraco/masterpages/default.master"
AutoEventWireup="true" %>
<asp:Content ContentPlaceHolderID="ContentPlaceHolderDefault" runat="server">
<umbraco:Macro Alias="PagedContent" runat="server">
</asp:Content>
```

AjaxTemplate.master

Step 4: Wire it all together

The next and final step is to wire it all together. What we are going to do then is use a little bit of jQuery to tell all the pager links (I've assumed you've wrapped these in a div with the class of ".pager") within the paged content to, instead of reloading the whole page, just reload the paged content code block via ajax using the altTemplate (remember, this just contains the html for paged content block, nothing else).

```
<script type="text/javascript">

    $(function(){

        $("#page-content").on("click", ".pager a", function(e){
            e.preventDefault();
            var ajaxUrl = $(this).attr("href") + "&altTemplate=ajaxTemplate";
            $("#page-content").load(ajaxUrl);
        });

    });

</script>
```

script.js

And we are done. When you now click on the pager links, a request will be made via ajax to get the contents of the altTemplate, which, when it returns, will replace the previous paged content block with the html in the response.

You could of course fancy it up and show a nice progress loader while it's loading, but I'll leave that to you to figure out :)

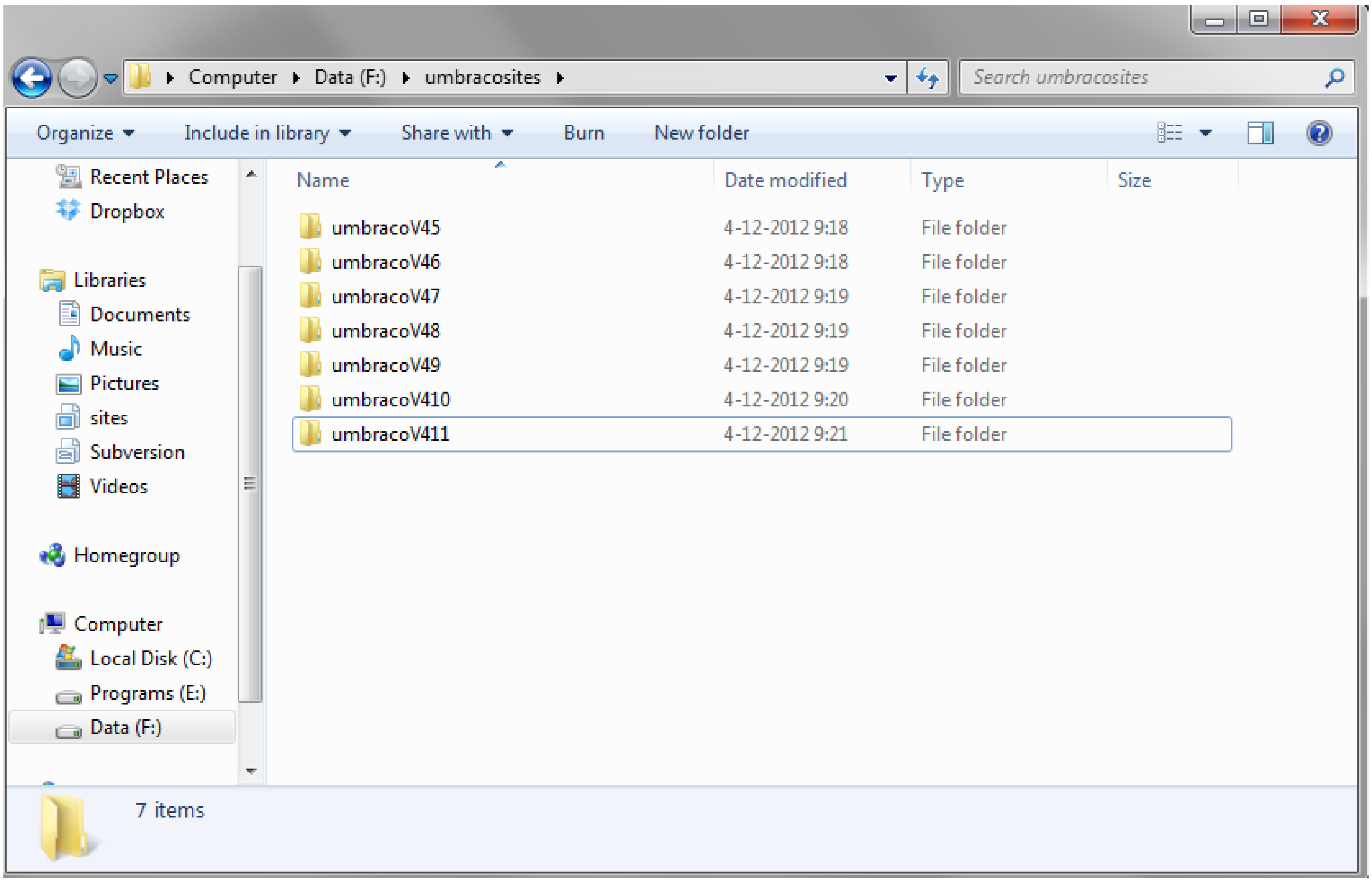
So there we have it, super simple, reusable, unobtrusive ajax paging using altTemplates.

Merry Christmas everyone.

Post build events for Package devs

— by Richard Soeteman

When building Umbraco packages I always use post build events in Visual studio to "deploy" the bits to my Umbraco development environments. Since most of my packages support multiple versions of Umbraco I need to test the package on every supported version of Umbraco. For every Umbraco version I have a complete website available in my Umbracosites folder.



Name	Date modified	Type	Size
umbracoV45	4-12-2012 9:18	File folder	
umbracoV46	4-12-2012 9:18	File folder	
umbracoV47	4-12-2012 9:19	File folder	
umbracoV48	4-12-2012 9:19	File folder	
umbracoV49	4-12-2012 9:19	File folder	
umbracoV410	4-12-2012 9:20	File folder	
umbracoV411	4-12-2012 9:21	File folder	

And I'm using post build events to copy the package files to the Umbraco websites.

```
XCOPY "$(ProjectDir)bin\CMSImport*.dll" "F:\umbracosites\umbracoV47\bin\" /y
XCOPY "$(ProjectDir)bin\umbraco.licen*.dll" "F:\umbracosites\umbracoV47\bin\" /y
XCOPY "$(ProjectDir)controls\*.ascx"
"F:\umbracosites\umbracoV47\umbraco\plugins\CMSImport\usercontrols\" /y
XCOPY "$(ProjectDir)pages\*.aspx" "F:\umbracosites\UmbracoV47\umbraco\plugins\CMSImport\pages\" /y

XCOPY "$(ProjectDir)bin\CMSImport*.dll" "F:\umbracosites\umbracoV48\bin\" /y
XCOPY "$(ProjectDir)bin\umbraco.licen*.dll" "F:\umbracosites\umbracoV48\bin\" /y
XCOPY "$(ProjectDir)controls\*.ascx"
"F:\umbracosites\umbracoV48\umbraco\plugins\CMSImport\usercontrols\" /y
XCOPY "$(ProjectDir)pages\*.aspx" "F:\umbracosites\UmbracoV48\umbraco\plugins\CMSImport\pages\" /y

XCOPY "$(ProjectDir)bin\CMSImport*.dll" "F:\umbracosites\umbracoV49\bin\" /y
XCOPY "$(ProjectDir)bin\umbraco.licen*.dll" "F:\umbracosites\umbracoV49\bin\" /y
XCOPY "$(ProjectDir)controls\*.ascx"
"F:\umbracosites\umbracoV49\umbraco\plugins\CMSImport\usercontrols\" /y
XCOPY "$(ProjectDir)pages\*.aspx" "F:\umbracosites\UmbracoV49\umbraco\plugins\CMSImport\pages\" /y
```

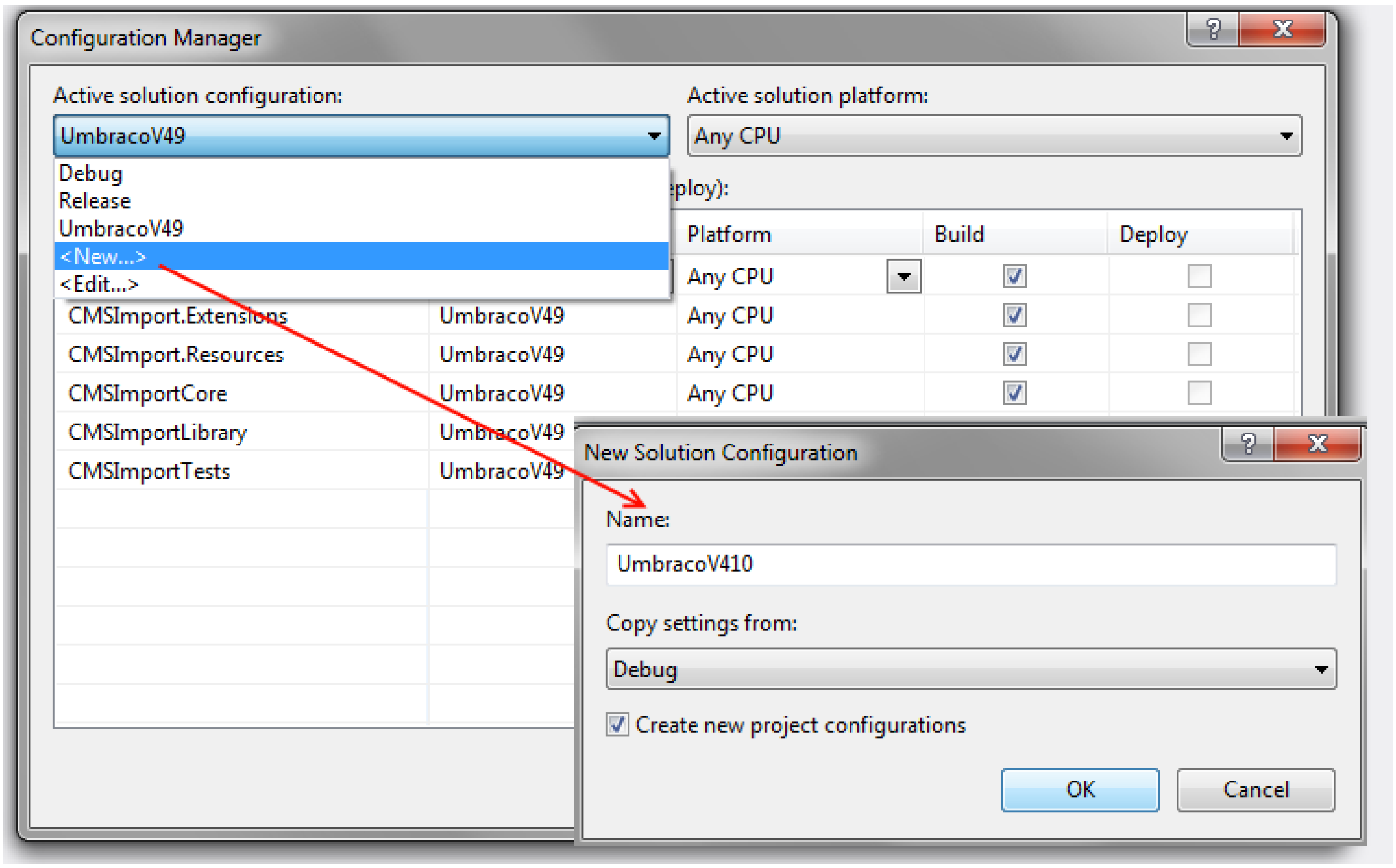
Post build events

Separating the package bits and websites allows me to completely destroy an Umbraco environment without having to worry about it, reinstall, or restore the environment when it is destroyed and deploy the bits from Visual studio and all is working again.

Improving this build process

Until recently I was very happy with this approach. A new version of Umbraco was released every few months, so every few months I needed to update the post build event. But then a new version of Umbraco was released every few weeks and I needed to update the post build event more often and when a new item is added to the build script I needed to copy this for every environment in the script. That got me thinking about a new way to do post build copies.

If you are a bit familiar with Post build event scripts you know that you can use variables in these scripts. One of these variables is the **$(ConfigurationName)** variable, which holds the value of the current Build configuration. By default it comes with two configurations **Debug** and **Release**. But when you go to Build, configuration manager and hit the Active solution configuration you see you can create your own.



When you build your project the $(ConfigurationName) will be replaced with the actual value of the configuration. So by creating a UmbracoV410 configuration I can update my post build script to the following and it will **replace $(ConfigurationName) with UmbracoV410** and files will be copied to **f:\umbracosites\** when using the UmbracoV410 build configuration. By creating configurations for every Umbraco version the package supports the post build event will work for every Umbraco environment.

```
XCOPY "$(ProjectDir)bin\CMSImport*.dll" "F:\umbracosites\$(ConfigurationName)\bin\" /y
XCOPY "$(ProjectDir)bin\umbraco.licen*.dll" "F:\umbracosites\$(ConfigurationName)\bin\" /y
XCOPY "$(ProjectDir)controls\*.ascx"
"F:\umbracosites\$(ConfigurationName)\umbraco\plugins\CMSImport\usercontrols\" /y
XCOPY "$(ProjectDir)pages\*.aspx"
"F:\umbracosites\$(ConfigurationName)\umbraco\plugins\CMSImport\pages\" /y
```

Optimized Post build events

Hope this helps you make it easier to test packages when developing for multiple versions of Umbraco!

Merry Christmas!

How to extend the back-office...

— by Lee Kelleher

... without touching the Umbraco core!

As the winter nights draw close and the days grow shorter, Umbracians around the world ponder the age old question... how can I customise the Umbraco back-office? *(Well maybe not, but it is a nice lead-in.)*

From a small tweak, to a major UI overhaul, with a little .NET code, it is possible to extend the back-office without modifying the Umbraco core source-code!

Show me, show me

All of the editor pages within the back-office, *(e.g. the ones in the main right panel area)*, expose two events (`Init` and `Load`) that can be hooked into. We'll focus on the `Init` one for now, here's what it looks like:

```
umbraco.presentation.masterpages.umbracoPage.Init += new MasterPageLoadHandler(umbracoPage_Init);
```

The best place to hook into such an event is by creating a .NET class, (either in your `~/App_Code` folder, or in a separate assembly), that inherits from the `ApplicationStartupHandler` (in versions prior to Umbraco 4.8 you can use `ApplicationBase`):

```
public class MyPlugin : umbraco.businesslogic.ApplicationStartupHandler
{
    public MyPlugin()
    {
        umbracoPage.Init += new MasterPageLoadHandler(umbracoPage_Init);
    }
}
```

> ***Tip:** If you are using Visual Studio, you can use `TAB+TAB` auto-complete to generate the following stub `EventHandler` method.*

Once you have hooked into the editor page's event, you can then associate a method (e.g. the `EventHandler`) to access the page itself.

```
protected void umbracoPage_Init(object sender, EventArgs e)
{
    // the fun starts here
}
```

The method's first parameter, although marked as an `object` is actually the instance of the `umbracoPage`. You will need to cast accordingly:

```
var page = sender as umbracoPage;
```

Okay, so now you have access to the editor page object... **now what?**

Snow me, snow me

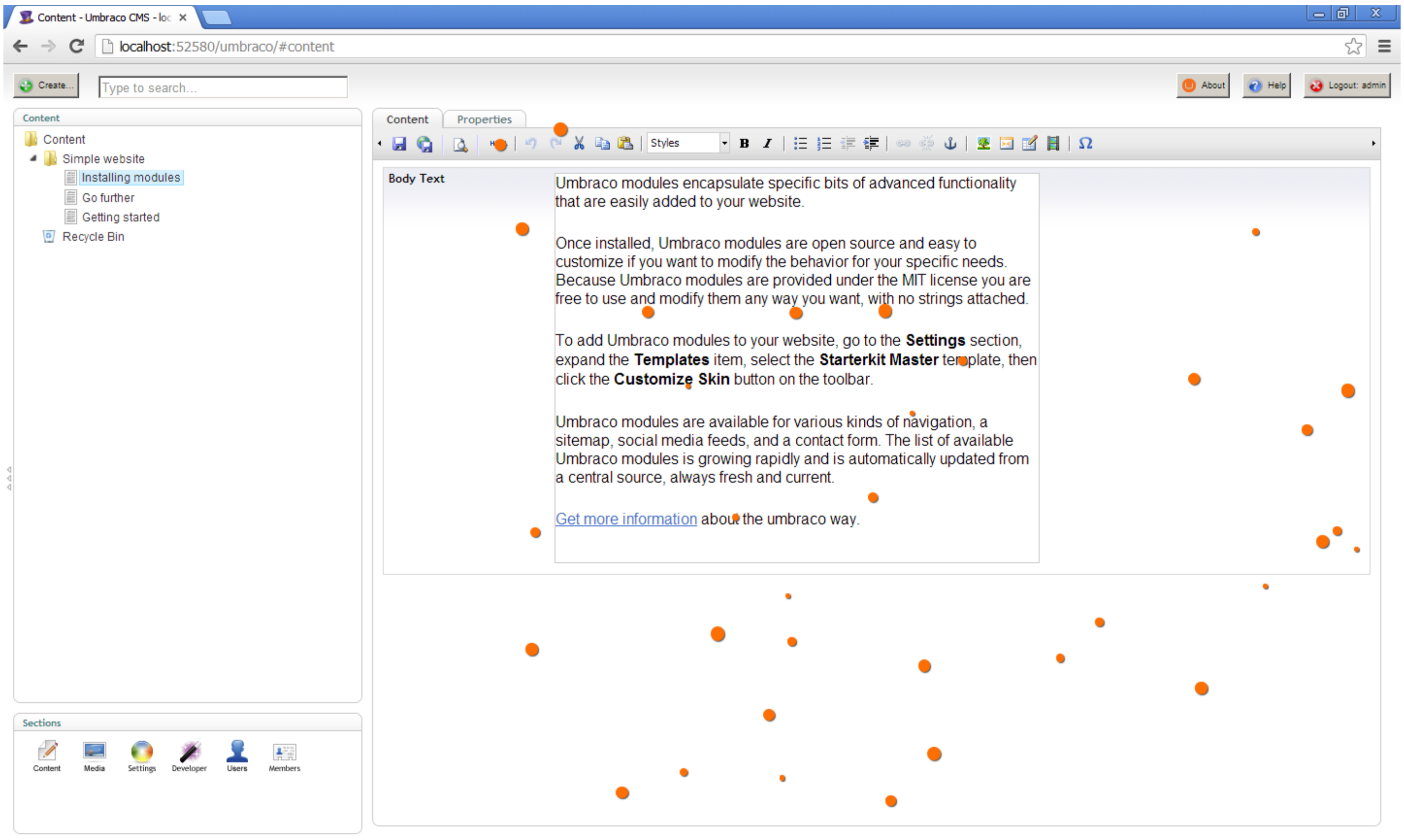
Since the festive season is nearly upon us, why not sprinkle a little Umbraco-themed winter snow on your content editors? ;-)

Using a jQuery plugin called Snowfall, we can inject the JavaScript into the `editContent.aspx` page (via the built-in ClientDependency framework).

Let's walk through the code to show you how; see inline comments for guidance:

```
protected void umbracoPage_Init(object sender, EventArgs e)
{
    // make sure the page is an umbracoPage; otherwise exit
    var page = sender as umbracoPage;
    if (page == null)
        return;

    // check that the path is allowed
    var path = page.Page.Request.Path.ToLower();
    if (!path.Contains("editcontent.aspx"))
        return;

    // attempt to find/create the ClientDependency loader for the page
    bool created;
    var loader = UmbracoClientDependencyLoader.TryCreate(page, out created);
    if (loader != null)
    {
        // set the custom path for your plugin
        loader.AddPath("SnowfallJs", IOHelper.ResolveUrl("~/scripts/snowfall/"));

        // load the JavaScript resources
        loader.RegisterDependency(998, "snowfall-loader.js", "SnowfallJs",
ClientDependencyType.Javascript);
        loader.RegisterDependency(999, "snowfall.min.jquery.js", "SnowfallJs",
ClientDependencyType.Javascript);
    }
}
```

Giving the desired result...



Umbraco modules encapsulate specific bits of advanced functionality that are easily added to your website.

Once installed, Umbraco modules are open source and easy to customize if you want to modify the behavior for your specific needs. Because Umbraco modules are provided under the MIT license you are free to use and modify them any way you want, with no strings attached.

To add Umbraco modules to your website, go to the **Settings** section, expand the **Templates** item, select the **Starterkit Master** template, then click the **Customize Skin** button on the toolbar.

Umbraco modules are available for various kinds of navigation, a sitemap, social media feeds, and a contact form. The list of available Umbraco modules is growing rapidly and is automatically updated from a central source, always fresh and current.

Get more information about the umbraco way.

Orange snowfall in the Umbraco back-office

The source-code is available to download here: `umbraco-snowfall.zip`

...and with that, I leave you with the tender words of Shakin' Stevens:

> **Snow is falling, all around me. Children playing, having fun.**
> **It's the season, for love and understanding.**
> ***Merry Christmas everyone!***

Community engagement

— by Jan Skovgaard

Hi all!

I'm Jan and I work as a frontend developer at Kraftvaerk where we create Umbraco solutions that provide great value for the client and their clients.

Today's post will not contain a single line of code - it will be about the cornerstone of Umbraco: The community!

Without this amazing global community that Chief Unicorn (aka Dear Leader (aka Niels Hartvig)) has been able to build over the years, Umbraco would not be at the place where it is today. The community is the true power behind the Umbraco project. Currently there are more than 30.000 active users on **Our.**

It's also because of this community that Umbraco has been able to survive after the decision to kill Umbraco 5 was announced at Codegarden 12 (CG12), which had been promoted with all kinds of talks about all the cool stuff that could be done in V5. I must admit that I was relieved when the decision was taken at the CG retreat, even though I was probably one of the very last to accept that it was the right move, out of concern about the future of Umbraco.

During the development of V5 it was as if the community was holding its breath in excitement to see what the brilliant developers in the HQ released next, rather than being much more involved in the project. Somehow V5 turned into a HQ project and the HQ and the community disconnected. It was like the community was running out of breath...

The turnaround

But what an amazing turnaround we've seen since CG!

Umbraco is now more amazing than ever! The old codebase is slowly being changed to the better and the best ideas from V5 are progressively being implemented in the different releases of Umbraco.

The community is breathing again, people are creating packages, even local user groups are collaborating about nice packages to enhance the back-office with new options, and people are participating in the various discussions about core development and the new back-office UI project codenamed "Belle".

Local user groups around the world have even had hackathons where they have focused on eliminating some of the minor bugs that on a daily basis can be a major pain for a developer or an Umbraco editor.

At the end of the year we will hopefully see the conception of Umbraco 6, which primary feature is a new public API that should be easier to work with. Who would have believed all this progress about 6 months ago? #H5YR community!

This really shows that the community unite to make Umbraco the best it can possibly be.

The future of Umbraco is looking bright for sure. But there is always room for progress and things that can be done smarter and better.

...So how can You contribute?

There are many ways to contribute to the Umbraco project:

Our
You don't have to be a hardcore developer in order to be able to make a difference. You can contribute by helping people out on Our, if you see someone having a problem you just found the solution for last week. Get in there and share the solution. You actually also contribute by asking questions... so go ahead and ask if there is something you're in doubt about, whether it's related to the Umbraco core or some of the many great packages for Umbraco. I'll guarantee that you will get a friendly answer from some clever minds!

Bughunting
If you find a bug you'll be doing us all a favour by submitting it to the Issue tracker

where the ever friendly Sebastiaan Janssen will figure out if something similar has been reported earlier, or if it has already been fixed. But let's not give the poor guy too much work - do him a favour and check if the issue you experience has already been reported. In that case, you can just vote it up. By voting you make it visible that this is something that should be taken care of. Before posting a bug you can also create a post on Our, or maybe even tweet to figure out if others have experienced the bug.

Core contribution
So if you're one of the hardcore .NET developer guys that know their way around in the core, or you think you can easily figure it out, you should of course be a core contributor. Then go ahead and create a fork of the source code and start driving HQ nuts with all your pull requests. Remember to keep an eye on the dev mailing list as well.

Documentation
This has always been known as the weak spot of Umbraco. And even though it's all become a lot better, people still complain about the lack of documentation. At some point I think the complaints are valid. But once again, we all have a responsibility to help document our lovely CMS. Therefore you can now grab your own fork of the documentation project and start writing nice documentation to the benefit for all of us!

Belle contribution
Finally there is an Umbraco project where frontend developers can show off their skills and contribute. The new interface is of course also open sourced. So go fetch the source for Belle and give it a spin. Remember to raise your voice in the debate about Belle too.

Host a hackathon
You can make an initiative to host a hackthon somewhere in your local community where the agenda can be to fix bugs, or perhaps work on a new killer feature for Umbraco. It could also be a hackathon on improving Our. If you're around Copenhagen on December 18, then sign up for the Our hackathon.

User groups
Create a user group and host some events. The kind of events you can host varies a

lot. Some user groups meet up at a pub in the evening once a month to have a nice chat and perhaps show off some cool stuff they've been brewing on. A user group can also host a bigger *Umbraco Festival*, which is usual a whole day of interesting Umbraco topics where local people and people from abroad can attend. So in short, the user groups don't need to host big events, they just need to gather some people and have a good time while discussing or playing around with Umbraco.

Blog
If you have found a cool way to achieve something using the Umbraco API, Razor, XSLT or how to build good IA in the content section, please do blog about it if you run a blog already.

Twitter
If you're on twitter make sure you keep an eye out for the #umbraco hashtag to see what people are writing about Umbraco. Some are looking for help, some are praising and a few are ranting. Also keep an eye on #umbLaunch, which is used when you have just launched an Umbraco site and want to brag about it. (It might give you some #H5YR in return! Yiiipeee! :)). If you attend CodeGarden you need to know about the #CG13 and #cgrumors tags as well. The latter is usually first used when it's bingo time! :)

I'm sure I probably missed out on some of the different ways to contribute in the above so feel free to notify me in the comments if you feel something should be mentioned.

So there really is no excuse for not engaging in the evolution of Umbraco, right? :)

That's it folks, remember to have fun and be friendly! Looking forward to seeing you all again at CG13.

Merry xmas and happy holidays everyone.

DAMP Gallery

— by Jeroen Breuer

I'm Jeroen and I work as a developer at Digibiz and Tribal Internet Solutions. At CodeGarden 12 I showed how I use DAMP in my own projects. I explained how everything works, but I couldn't share any code because I used a customer project as an example. Because Umbraco now supports MVC, I thought it was about time to create a website which shows how you can use DAMP in the new view templates. That why I'm introducing the DAMP gallery. It's a complete website that can be used as an example. It runs on SQL CE so everything should work out of the box.

You can view a live example here.

You can setup this example in a few easy steps:

1. Download the zip at the bottom of this article.
2. Unzip the folder.
3. Open the Visual Studio project.
4. Run the website from within Visual Studio.
5. Umbraco login user: admin - password: damp

The website shows how DAMP can be used for an image and video gallery. It also has a little MNTP bonus, just to show how that can be used in 4.11. It has paging and on the gallery 1 page I also use Matt Brailsford's altTemplate paging example. I even extended it a little to support deeplinking.

The DAMP gallery uses multiple packages which are all related to the media section:

- Digibiz Advanced Media Picker: It uses a version which is newer than DAMP 2.0. Change Set 2aff58e42dca from CodePlex is used. This version supports SQL CE and has better performance.
- DAMP Property Editor Value Converter: Makes it easier to use DAMP in Razor.
- Create Crops: Ensures that all crops are available.
- Track Media: Each time an image or a video is clicked it will be tracked by Google Analytics.

- Multiple File Upload: A custom version which supports custom media types. The uploader introduced in 4.9 is disabled because it doesn't support custom media types.
- uTube: Only used for the uTube Single Video Picker datatype on the YouTube custom media type.
- Restricted Upload: Used on the Video custom media type so only mp4 files can be uploaded.

Because of the DAMP Property Editor Value Converter package the Razor code looks very clean. This is the complete Razor file to show the gallery on the gallery 1 page. This example is dynamic, but there is also a strongly typed example in the website source code.

```
@inherits Umbraco.Web.Mvc.UmbracoTemplatePage
@{
    dynamic i = CurrentPage.images;
    if (i.Count > 0)
    {
        var p = Helper.GetPager(6, i.Count);
        <ul class="photogallery">
            @foreach (dynamic d in i.Skip(p.CurrentPage * p.ItemsPerPage).Take(p.ItemsPerPage))
            {
                <li>
                    <a class="popup" rel="gallery" href="@d.Url">
                        <img src="@d.Crops.gallery" alt="@d.Alt"/>
                    </a>
                </li>
            }
        </ul>
        <div class="pagination">
            @foreach (var number in p.Pages)
            {
                if (number - 1 != p.CurrentPage)
                {
                    <a href="?page=@number">@number</a>
                }
                else
                {
                    <span>@number</span>
                }
            }
        </div>
    }
}
```

The video below also shows how you can use the DAMP Gallery.

Online Video Clip

I'd like to thank Ralph van Vugt who helped me with the HTML.

Download the DAMP Gallery source code

Give me JSON

— by Thor Madsen-Holm

As a frontend developer I love Umbraco because of the fact that I can get a very long way without ever touching any .NET code. In the last year or so I have been doing more and more work with AJAX, be it singlepage apps or just parts of a website that loads content asynchronously.

After loading content, I usually want to render it as HTML on the page. To do this, I create an altTemplate (see Matt's previous post about altTemplates, if you want to know more) and insert an XSLT macro on that page.

In the past I usually had this macro render all the HTML I needed. Then when I requested the altTemplate, I could just inject the contents into the page like this:

```
$.ajax({
    url: "alttemplateUrl",
    dataType: "html",
    success: function(html){
        $("#insertDataInThisElement").html(html);
    }
});
```

And voila! New content is rendered on the page. This works great when you are working with smaller chunks of HTML and don't need to use data from the request that's not intended to be rendered.

So what can we do if we want something that's easier to work with in JavaScript than just an HTML string? We use JSON of course! But hey, how do I get Umbraco to output JSON without having to do any .NET work? If you are an XSLT man like I am, you simply install Peter Duncanson's brilliant package XSLToJSON (Which also works in Umbraco 4.11.1)

The package creates a `toJSON.xslt` macro and a JSON template. The XSLT macro outputs `$currentpage` as JSON, like this:

```
<xsl:template match="/">
    <xsl:value-of select="orc.XSLToJSON:XmlToJson($currentPage)" />
</xsl:template>
```

See it in action by going to **yourwebsite.com/JSON** to see what it outputs.

You could extend this template to output a specific page by doing something like this:

```
<xsl:variable name="pageId" select="umbraco.library:Request('pageid')" />

<xsl:template match="/">
    <!-- Make sure pageId is not empty -->
    <xsl:if test="normalize-space($pageId)">
        <!-- Get the page -->
        <xsl:variable name="page" select="umbraco.library:GetXmlNodeById($pageId)" />

        <!-- Output the page as JSON -->
        <xsl:value-of select="orc.XSLToJSON:XmlToJson($page)" />
    </xsl:if>
</xsl:template>
```

Then you could get a specific page via JavaScript like this:

```
$.ajax({
    url: "/JSON?pageId=1048",
    dataType: JSON,
    success: function(data){
        // Do something with the data
    }
});
```

Now we could use a templating library such as handlebars.js to render the content on an HTML page.

If we do something like the example above, there is a good chance we won't need all the data from `$currentPage`. Suppose we just need the `title`, `bodyText` and an image, we don't want to use precious bandwidth to load unnecessary content. So how can we get around that? We construct our own XML and pass it to the `XmlToJson()` extension like this:

```
<xsl:variable name="pageId" select="umbraco.library:Request('pageid')" />

<xsl:template match="/">
    <!-- Make sure pageId is not empty -->
    <xsl:if test="normalize-space($pageId)">
        <!-- Get the page -->
        <xsl:variable name="page" select="umbraco.library:GetXmlNodeById($pageId)" />

        <xsl:variable name="customXml">
            <root>
                <!-- We use normalize-space() in the apply-templates statement to ensure we only
get elements that have content -->
                <xsl:apply-templates select="$page/title[normalize-space()]" />
                <xsl:apply-templates select="$page/bodyText[normalize-space()]" />
                <xsl:apply-templates select="$page/image[normalize-space()]" />
            </root>
        </xsl:variable>

        <!-- Output the page as JSON -->
        <!-- Because we constructed the xml ourselves we need to use msxml:node-set() to convert it
```

```
to a nodeset -->
        <xsl:value-of select="orc.XSLToJSON:XmlToJson(msxml:node-set($customXml))" />
    </xsl:if>
</xsl:template>

<xsl:template match="title">
    <title>
        <xsl:value-of select="." />
    </title>
</xsl:template>

<xsl:template match="bodyText">
    <bodyText>
        <xsl:value-of select="." disable-output-escaping="yes"/>
    </bodyText>
</xsl:template>

<xsl:template match="image">
    <xsl:variable name="imageXml" select="umbraco.library:GetMedia(., 0)" />

    <!-- It wouldn't be of much use if we only returned the media id, so we get the url -->
    <imageUrl>
        <xsl:value-of select="$imageXml/umbracoFile" />
    </imageUrl>
</xsl:template>
```

Now when we request the altTemplate with a pageid as parameter, like this: yourwebsite.com/alttemplate**?pageid=1048**, we only get the `title`, `bodyText` and `imageUrl`.

Here's the JSON output from a random page I created:

```
{
   "root":{
      "title":"OMG! This is JSON",
      "bodyText":"Umbraco modules encapsulate specific bits of advanced functionality that are
easily added to your website.",
      "imageUrl":"/media/35/bender_bending_rodriguez.jpg"
   }
}
```

And that's how you can get JSON data from an altTemplate with the use of the XSLToJSON package.

I'm no Razor expert, but I'm sure you can do something similar there.

That's it, now go make something awesome with JSON from Umbraco :)

Happy holidays!

Optimising Umbraco for speed

— by Pete Duncanson

Howdo. Thanks for stopping by. I thought I'd do a bit about two quick tricks you can do to make an Umbraco site go super fast: Macro caching and image compression. Nothing new but optimisation keeps cropping up when I talk with you guys at the various events.

Before we start ask yourself if the site even needs any optimisation. I'm a big fan of only doing work if required. If it's a small site with a few macros then it simply might not be worth doing anything. Remember the 80/20 rule and you should be fine.

Macro Caching

First up macro caching. Every time you render a page you cause the macros you've added to that page to run and generate out the HTML we send down the wire. The "thinking time" or more correctly CPU cycles the server needs to do that can be a bottle neck, why spend time working out the menu navigation links each and every page load? It does not change all that often right? So why not work it out once and then save the result and reuse that on repeat request. That's what macro caching does and it's built into Umbraco, you just have to switch it on. We tend to overlook this bit while developing though as we have kick ass machines and we are the only one hitting it.

Warning: Switching on macro caching while you are still developing is not a wise move. At best it can simply slow you down, at worst it can lead to some serious head scratching/violence like so:

Online Video Clip

while you try to work out why your changes are not appearing on the site. Stay sane, only add it to a staging/live production sites.

Check your macro render speeds. You can get a whole heap of debug information out of Umbraco by switching on the "Debug and Trace mode" by adding `?umbDebugShowTrace=true` to the end of your URL, or install the Chrome Umbraco Developer Extension which is really really handy (careful this will only work as long as you have not disabled debugging on the site, by default this is switched on so you know).

Protip: In your sitewide CSS add this rule to make it easier to read the trace output no matter what your background colour is:

```
#__asptrace {
  background-color:white;
}
```

CSS trick to ensure your trace statements are easily readable regardless of your site's design

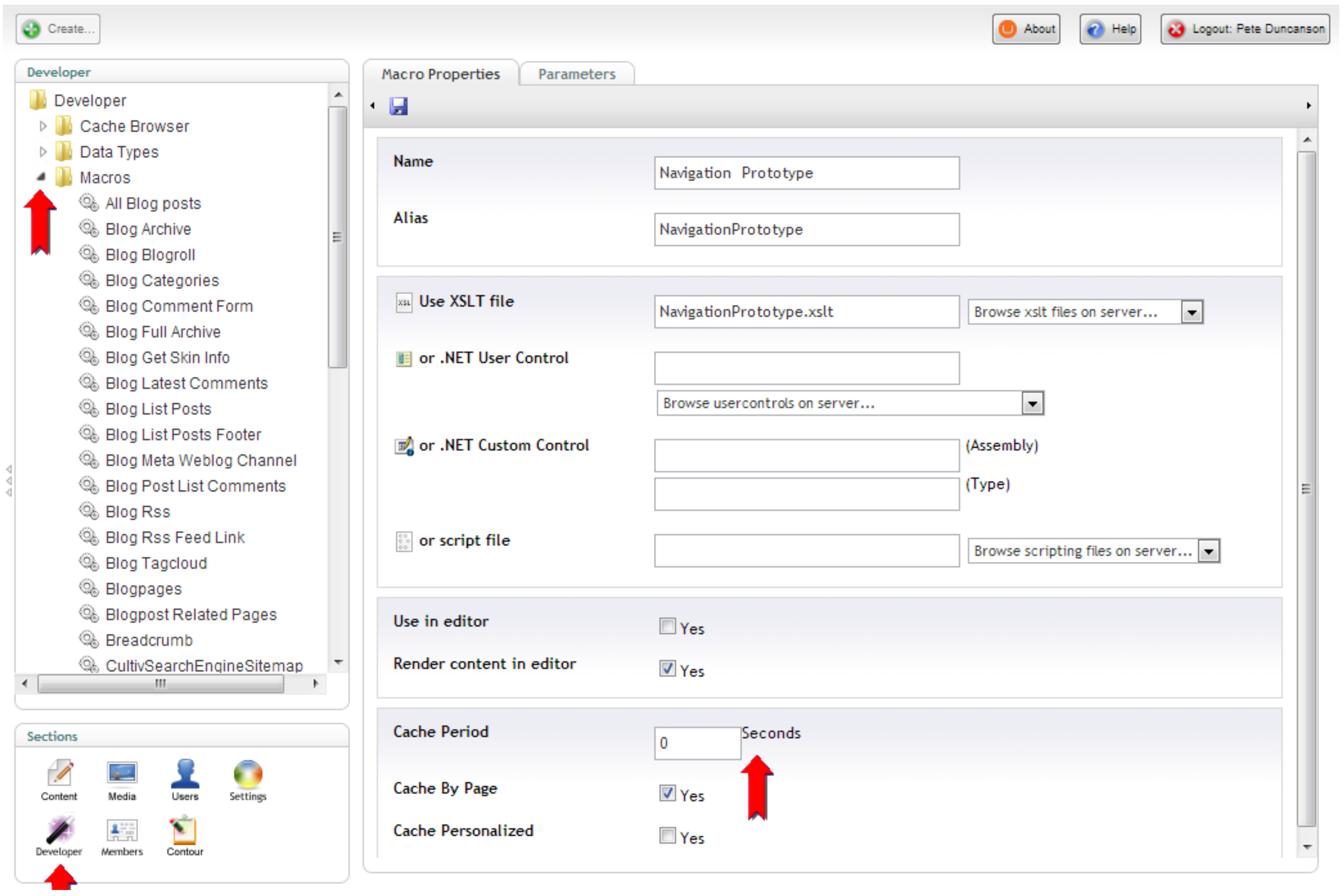
I've got a whole raft of tips on tidying up trace output but that will have to wait for another day/post, remind me :)

Now at the bottom of the page you should have a huge ugly table of debug output including some lines about Macro render times. There should be a total time up to that point (the 3rd column) and time for that action (the 4th column). Keep an eye out for any macros that are slow (slow to me is anything over 1/10th of a second), we will hit these ones first.

The idea is to hit the slowest macros first, switch on caching for that one, refresh to see the effect and then do the next slowest. The slowest macros should be the ones that are doing the most "work" or are coded really badly...either way caching will help (but try to fix any slow running code first).

Be careful what you cache, it can be really tempting to go through all the macros and set them all to cache. You don't want to cache search result pages for instance (been there, done that). Also, if a macro is rendering in 1/100th of a second it's not really a snail now is it? Leave it be unless you are super keen or bored with having a life outside of work. An exception to that is if the site only has a fraction of the data in it (let's say it's just got a bit of dummy content in it) then you might not be seeing "true" render times that the live site might experience. Another good reason to only really setup macro caching on a live/staging site.

So to get to macro settings you need to go to the **Developer > Macros** and select your slow macro. If you can't see the Developer section then you don't have access (if you're a client, stop trying to mess with the site, you'll break something!).



Name: Navigation Prototype
Alias: NavigationPrototype
Use XSLT file: NavigationPrototype.xslt — Browse xslt files on server...
or .NET User Control: — Browse usercontrols on server...
or .NET Custom Control: (Assembly) (Type)
or script file: — Browse scripting files on server...
Use in editor: ☐ Yes
Render content in editor: ☑ Yes
Cache Period: 0 Seconds
Cache By Page: ☑ Yes
Cache Personalized: ☐ Yes

Macro caching settings, important bits highlighted

The important setting is "Cache Period", by default this is set to zero which means don't cache. The number here is the number of seconds you want to cache the output for. Simply change this to a valid number (try 10 seconds for now), leave the other settings as is ("Cache by page" checked and "Cache Personalized" unchecked), save the macro settings and that's it. The macro will now cache itself the next time it runs and won't refresh itself until the number of seconds you added have passed. Reload your page and you should see next to no difference because it still has to generate the output the first time, reload again and now the output will be pulled from cache super fast!

How long to cache for? Depends really. Any caching can make a huge difference. You can go the route of micro caching for just a few seconds, or if the content really does not change that much, you can bump the time up some more. Even quiet sites can benefit; headers, navigation etc. will be re-used during a user's visit. They might be the only visitor that day but each page they visit after the first one will have cached content so will appear faster.

Caching makes the site faster and reduces load on the servers (especially important on a busy site). We are trading memory space for CPU cycles here so make sure your site has enough memory to cope, IIS will handle the memory for you but to make the most of caching you should have enough memory to store all the different output in. The cache will expire if the site is not hit for 15 minutes (as IIS shuts down the app pool to save resources) or if the app pool recycles for some reason. You can get around this by polling the site every 10 minutes or similar from another server/service but ask yourself if that's really all that important/required before doing it defacto.

Compress your images

Another nice simple mod you can do is to compress your images. This is easy enough to do when building the site but harder when you hand the site over to clients who start uploading HUGE jpg files from their digital cameras or that they have found somewhere in an uncompressed state. All that hard work ruined by those pesky clients.

ImageGen to the rescue as ever. This awesome image resizing script ships with Umbraco so you can use it on any build. The freebie version is up to the job too (however, if you've ever used ImageGen and not bought at least one Pro licence, you really ought to think about it on the next project, Doug deserves it). You don't have to use it just to resize images though, you can also use it to compress them.

ImageGen resizes images based on the values you pass in on the query string and out of the box does a great job of compressing images too, but you can tweak this a little more if you want. There is a little known setting called "Compression" which can be used with jpg images.

This is the amount of compression as a percentage that you would like to apply to the image. The default is a rather high 90%. You can pass through a lower value however and make some really good savings on file size for very little loss of quality. The range of 72-80% seems to work for me, depends on the type of images your client mainly use.

```
(original, 611K)
http://offroadcode.com/media/14930/original.jpg

(90% compression by default, 156K)
http://offroadcode.com/umbraco/ImageGen.ashx?image=/media/14930/original.jpg

(75% compression, 140K)
http://offroadcode.com/umbraco/ImageGen.ashx?image=/media/14930/original.jpg&compression=75
```

Various URLs, can you spot the difference in quality?

By default always use ImageGen to render out any images that the client can change, it will help save a lot of bandwidth. Tweaking it via the compression option can save some more for an extra gold star.

Hope that gives you something think about and some pointers on where you could make some saves. There are plenty of other tips but these two are Umbraco specific which is what these posts are all about. Hope you enjoyed it, if so feel free to buy me a beer next time we meet.

Festive greetings to you all, my extended Umbraco family :)

Pete

Remember the Editors!

— by Kim Andersen

Hi there fellow Umbracians!

My name is Kim Andersen and I'm from Denmark, working for Kraftvaerk in Aarhus as a frontend developer.

In this post I'll be talking a bit about the back-office inside of Umbraco. Not about how to extend it or redesign it, but I'll be talking about how to give the editors and the webmasters a good experience. After all, they are the ones that are using Umbraco every single day.

So you won't see a single line of code in this post, and maybe you won't even learn anything new. Still, I think that you should read along :)

In my time working with Umbraco, I've seen a lot of implementations. A lot of good ones, but also a whole lot of bad ones.

A typical implementation focuses 100% on the frontend of the website. The stuff that the end-users are seeing. Of course they are also the ones to please, as they are our clients' customers.

Unfortunately, there's a lot of developers and consultants, that forgets the editors and the webmasters that are using Umbraco every day. The editors should have a good experience as well, otherwise they'll end up being frustrated and in the end they'll maybe talk bad about Umbraco. Either way, they will never ever recommend Umbraco to anyone again. And that's a shame if the reason is that they've had some bad experiences inside of the back-office, right?

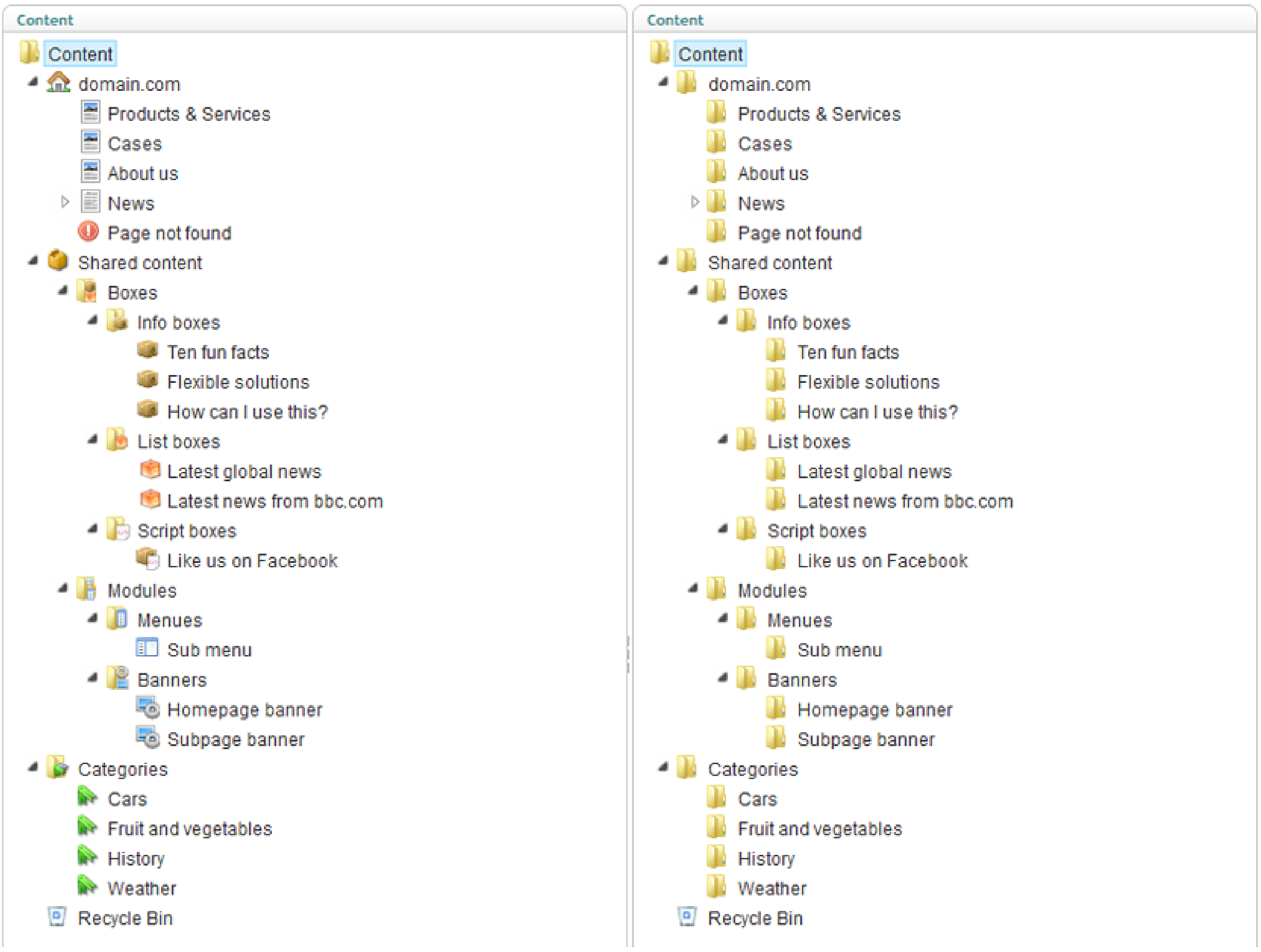
Actually, it doesn't take much effort to give the editors a good experience working with Umbraco. I'll say it's a combination of icons, descriptions and consistency. It's all about helping the editors as much as possible, and making sure that they can maintain and use their new CMS as easy as possible.

Icons and thumbnails

So, icons and thumbnails really helps the editors a lot - a whole lot! Those small icons can give the editors a nice and quick overview over their content, without thinking of every single page and document type.

In a lot of those implementations that I have seen through the years, the most used icon must be that ~~damn~~ folder icon - The default icon for new document types in Umbraco. The folder icon doesn't help the editors a lot, especially not if it's the icon on most of the document types. So if we have a site with several different document types, and all of them are using the folder icon, no one, not even the folks that have implemented it, can get an overview over the content.

Try to have a look at the following screenshot:



See the difference?

See what I'm talking about?

I know that you might think that it sucks to spend time selecting and creating icons and thumbnails for every single document type, but there are actually a lot of free icons out there. For example you can install the "FamFamFam icons"-package that gives you hundreds of free icons. That's a good starting point. The next step could be to find an icon or two and maybe combine those to a single icon, that symbolizes what content we're talking about on each single page type.

Descriptions

Descriptions on each property field and on each document type can also make a huge difference for the editors (and also for yourself when the customer suddenly needs help, new features etc. a year after the site was implemented).

Again you might thing that it's a waste of time describing every field on all document types, but if you are just making it a natural part of creating a property, then it doesn't take you many seconds to describe a property. Since I began doing this, there's not a single property in any of the solutions that I've implemented that doesn't have a description. Not even a basic title field on a sub page.

Consistency

This part can be a challenge for even a very skilled and experienced Umbraco developer. When I say consistency, I mean the way that the different document types are built and structured. If it's possible, then try using the same property names and tab names on different document types, if the functionality is sort of the same.

Create similar properties under the same tabs across document types, and sort them in the same order from document type to document type. That way the editors can easier find the fields they are looking for, instead of having to search around the different tabs for a property that's called one thing on one document type, but another thing on another document type.

Packages

I also want to mention a couple of packages for helping the editors even more.

First off, I want to mention the "Document Type Fieldsets"-package by Tom Fulton. This package gives you the ability to group the properties under the different tabs, giving the editors a better overview of the properties under one single tab.

Another package I want to mention is the "Attackmonkey Tab Hider"-package by Tim Payne. This package can help you personalize the back-office for each user by hiding selected tabs from specific users. A very great package as well.

How about Belle?

As you have probably heard, the back office will be redesigned in project Belle. This will of course affect some of the things that I've talked about in this post, but the overall thoughts will still be the same; Help the editors, and make it as easy as possible for them to find the right nodes, navigate the back-office, maintaining their content etc.

Conclusion and your thoughts

In the end it's a benefit for all of us whenever a new great Umbraco solution is implemented, but on the other hand it can also be a problem for all of us whenever a bad Umbraco solution is implemented.

I hope that you liked the post, and I would love to discuss the stuff that I've talked about, in the comments, so please give me your view on the topic. Do you think it's a waste of time? Are you already doing this stuff? Have I forgotten some important parts? Or do you have any other comments/questions, then please don't hesitate commenting below.

Merry Christmas Everyone.

Some extra ressources:

Here's acouple of extra ressources mentioned in the comments:

- Categories folder-icon

- Icon Picker package - (have more than one icon for each document type)
- DocTypeMixins package

On Being a Superhero

— by Douglas Robar

Admit it, you've always dreamed of being a superhero. And this year for Christmas I'll show you three quick and easy steps so that you too can be a mega-awesome acey-pacey brilliant-and-all Umbraco superhero. Your content editors will be singing your praises all season long.

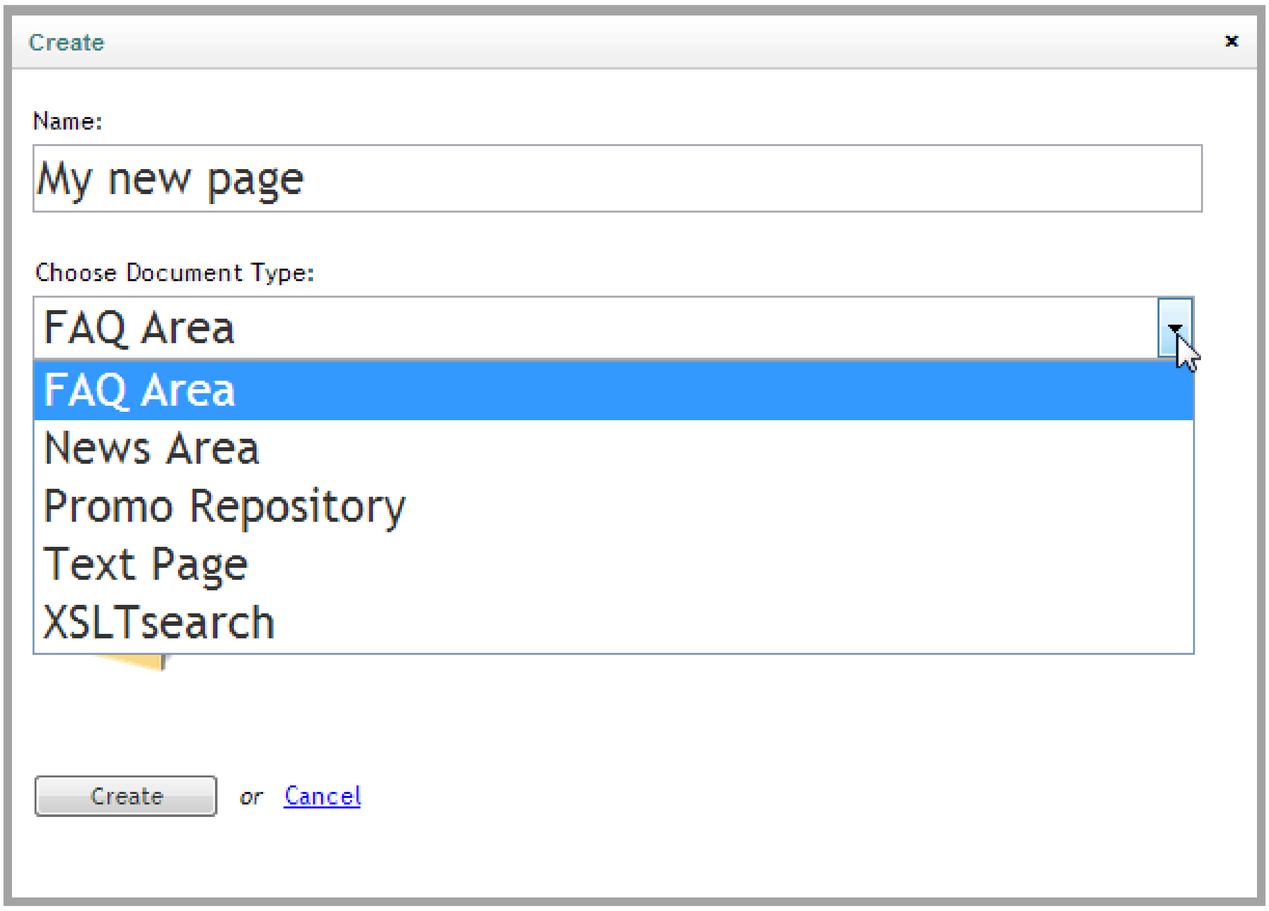
1. Document Types - a choice of one

What is it? Who are you? What do you want? ... My god you've gotten fat!

Ever had too many choices and been unsure which to select? What happens? Do you:

1. do nothing
2. pick something at random
3. accept the default and hope it's correct
4. take the time to figure out the best thing

Those same options often run through Content Editors' minds deciding which document type to use when creating a new page. Consider this dialog:

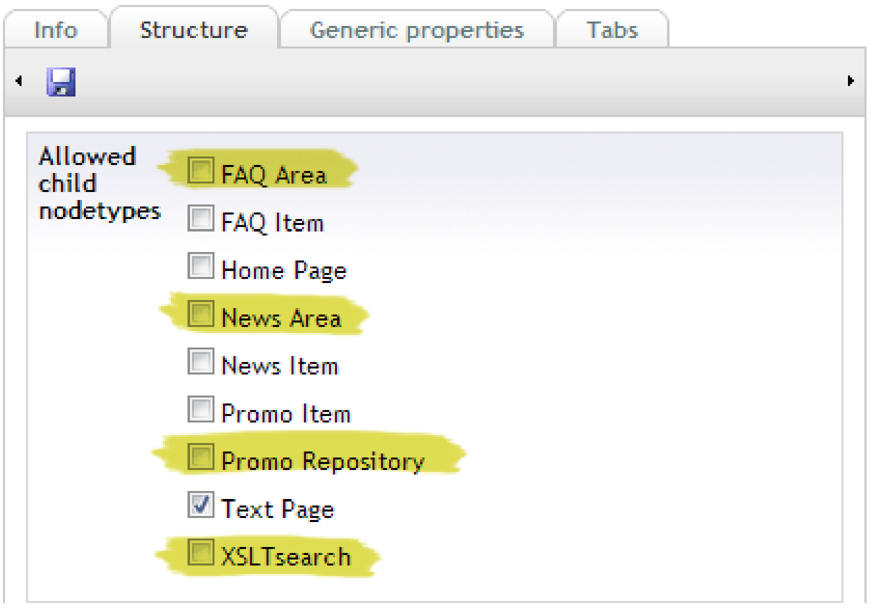


What should the user select?

You can see the problem. There are too many choices. Worse, because the document types are listed alphabetically, the first one may not be a good default choice.

If there were only *one* choice in the list then users would be bound to get it right. In fact, that's the reason that, when we create a News Area document type for example, we set the structure to only allow individual News Items below it. The content structure is made obvious and bullet proof. That's the power of one.

So do your users a favour and get your document types into shape. After you've made your one-of-a-kind pages in the content tree (such as the News Area, the search page, the FAQ section, the events area, the staff section, etc.) deselect them on the document types structure tab.



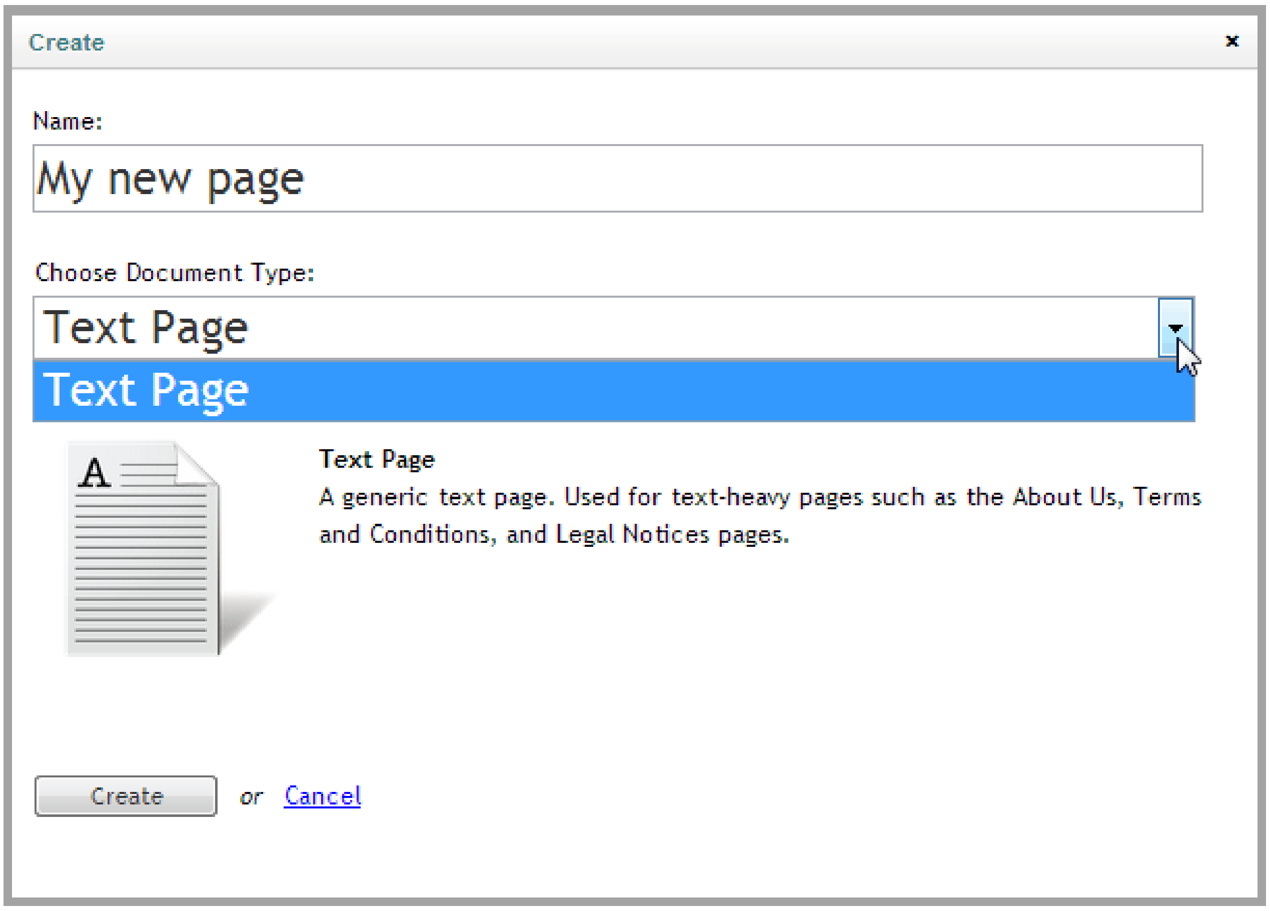
Info | Structure | Generic properties | Tabs

Allowed child nodetypes

- [ ] FAQ Area
- [ ] FAQ Item
- [ ] Home Page
- [ ] News Area
- [ ] News Item
- [ ] Promo Item
- [ ] Promo Repository
- [✓] Text Page
- [ ] XSLTsearch

DE-select one-of-a-kind document types after their content pages have been created

The result is elegant simplicity.



Create

Name: My new page

Choose Document Type: Text Page

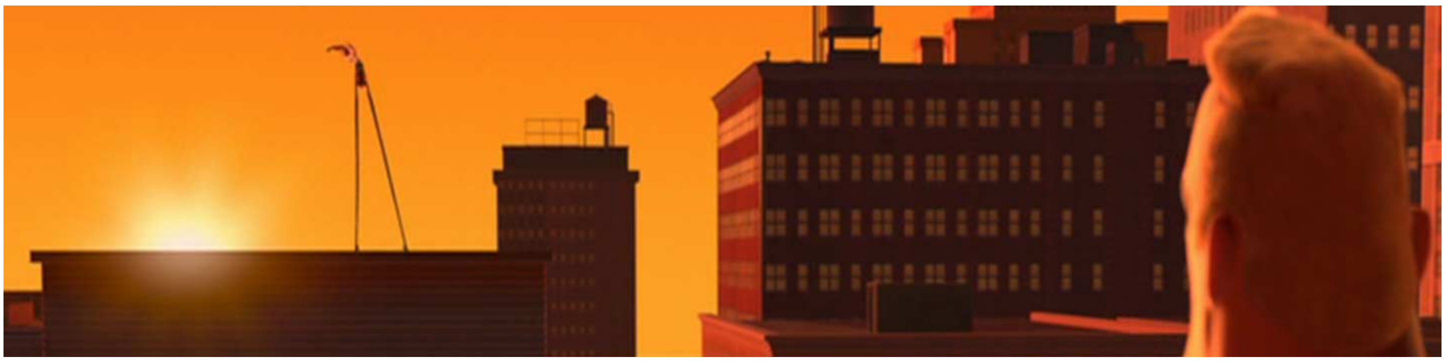
Text Page

A generic text page. Used for text-heavy pages such as the About Us, Terms and Conditions, and Legal Notices pages.

Can't go wrong with a choice of one!

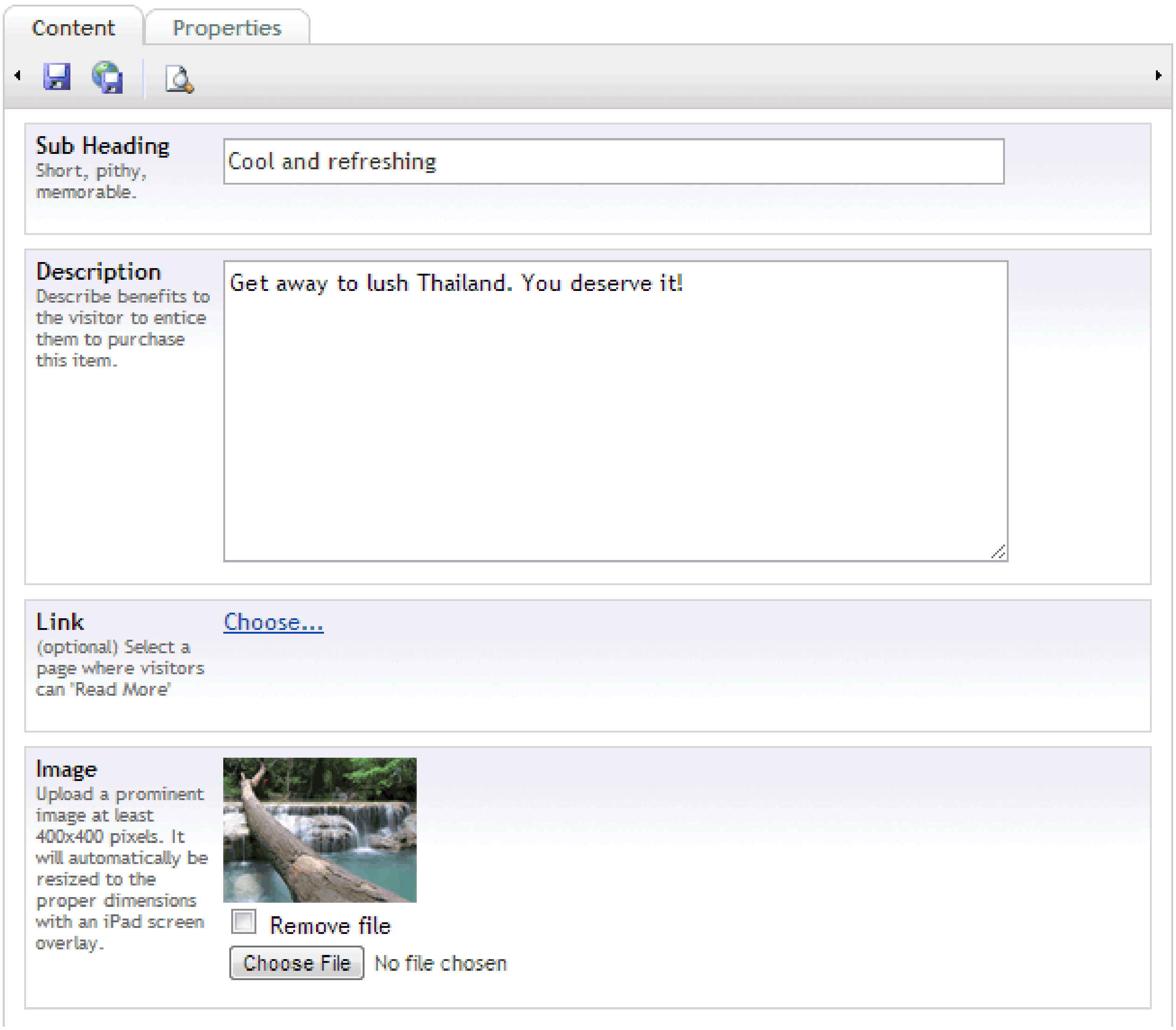
It's now impossible for users to accidentally create additional one-of-a-kind pages in the content tree. And as an added bonus they can't move or copy them either because that isn't allowed by the document type structure. Everyone wins.

2. Contextual Help


I think you need to be more... flexible!

You already add descriptive text to your document type properties to answer the "what should I enter into this field?" question that you'll otherwise inevitably get from content editors.

You *do* add descriptive text... right? If not, you're making much more work for yourself and your users. Kim explains why. By providing a bevy of perfectly-placed help text for fields you'll already be a hero to your users.



Sub Heading
Short, pithy, memorable.
Cool and refreshing

Description
Describe benefits to the visitor to entice them to purchase this item.
Get away to lush Thailand. You deserve it!

Link
(optional) Select a page where visitors can 'Read More'
Choose...

Image
Upload a prominent image at least 400x400 pixels. It will automatically be resized to the proper dimensions with an iPad screen overlay.
Remove file
Choose File No file chosen

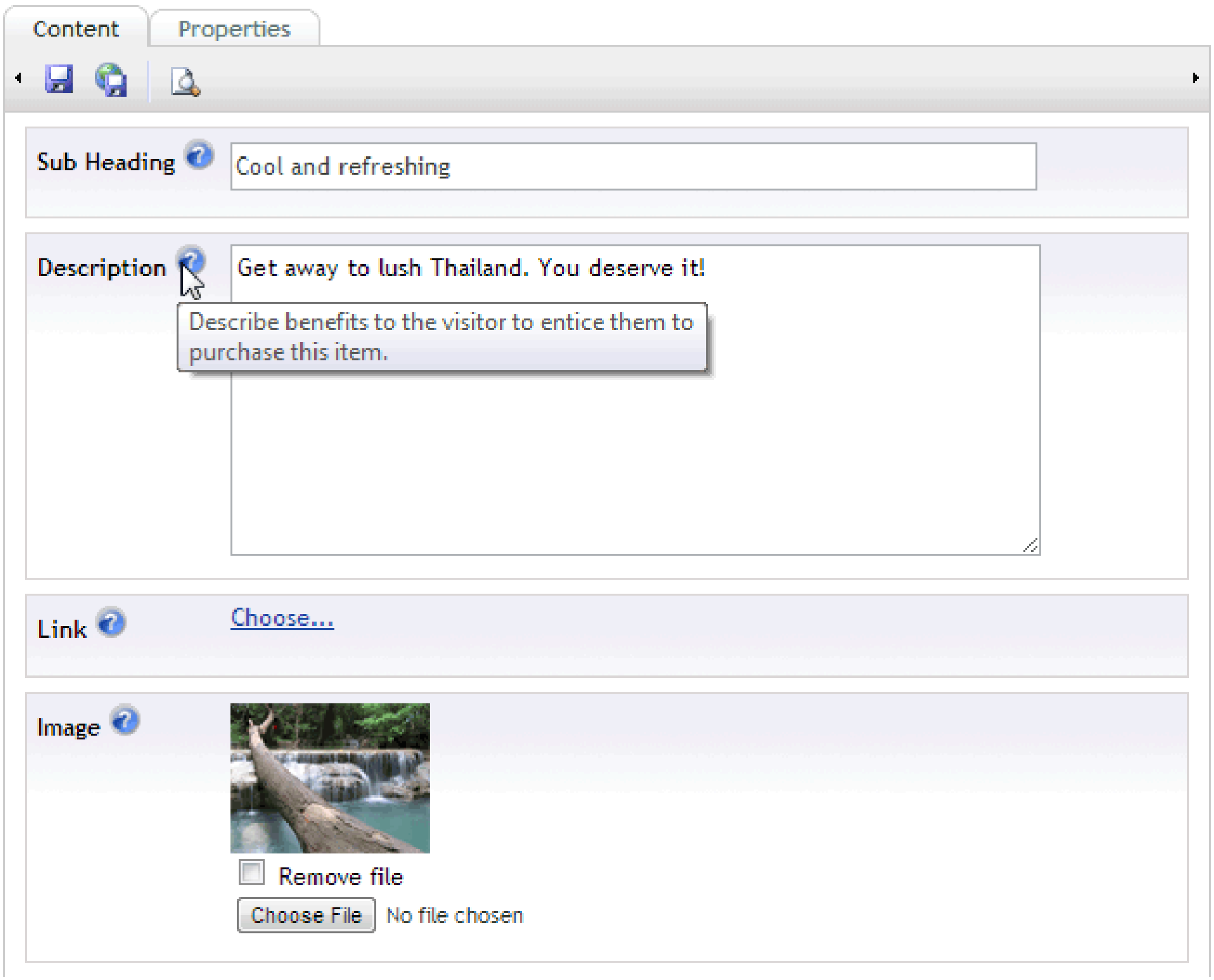
Descriptive text for properties makes you a hero

Now go one step further and become a superhero! A month or so after the site has gone live, when content editors are familiar with the fields and don't need as much in-your-face help any more, change to displaying a small help icon rather than the full text.

Change the `PropertyContextHelpOption` in the `/config/UmbracoSettings.config` file from '`text`' to '`icon`':

```
<!-- Show property descriptions in editing view "icon|text|none" -->
<PropertyContextHelpOption>icon</PropertyContextHelpOption>
```

Now, hovering over the icon displays the descriptive text in the tooltip. Still available when needed without being intrusive. As a bonus, the content entry screens are tighter and smaller, which editors prefer.



Sub Heading: Cool and refreshing

Description: Get away to lush Thailand. You deserve it!

Describe benefits to the visitor to entice them to purchase this item.

Link: Choose...

Image:

Remove file

Choose File No file chosen

Rollover icon to see the help text. It's right there when you need it.

3. Personalized Help Pages

Listen closely. I'd like to help you but I can't.

Did you know that Umbraco's help button at the top of the back office interface is context-sensitive? It will take you to a wiki page that deals specifically with the screen being displayed (see our.umbraco.org/wiki/umbraco-help).

But these generic help pages don't know about your site's special macros, content organisation, naming conventions, etc. Wouldn't it be great if you could have personalised help pages unique to each of your client sites? Merry Christmas, you can!

The basic idea is to create a site with pages that contain not only the generic details but also all the how-to instructions, conventions, screen shots, help videos, and more that the site uses.

You can make as many pages as you need for any combination of **application section** (content, media, settings, etc.), **page displayed** in umbraco (dashboard, editContent, etc.), **language** (translate your help as well when you have multi-lingual sites). and **user type** (administrators might see more technical details than writers).

Then update the help setting in the `/config/umbracoSettings.config` file to point to your custom help page(s).

```
<help defaultUrl="http://our.umbraco.org/wiki/umbraco-help/">
    <link application="content" applicationUrl="editContent.aspx"  language="en" userType="Editor"
helpUrl="http://www.site.com/{0}/{1}/{2}/{3}" />
    <!-- Result url: http://www.site.com/content/editContent.aspx/en/Editor -->
</help>
```

Create personalised help for Editors working in English when editing a content page

See our.umbraco.org/documentation/using-umbraco/Config-files/umbracoSettings for full details of the help syntax and more examples.

Hey presto, you've just given your users an amazing Christmas present! And solidified your reputation as an Umbraco superhero in the process.

Merry Christmas, everyone. Let loose your inner superhero!

Bulk-Assigning Media Items

— by Sebastian Dammark

Setting the scenario:

You've got a new client who has a major product catalog (~1500 products) and they've sent it to you in CSV, looking something like this.

```
prodid;prodname;proddesc;prodnumber
01;Giant Hat;This hat only fits dwarfs;GH001V023;
02;Red Hat;This hat only fits redheads;GH002V458;
```

The product catalog data in CSV format

The data itself we import with CMS Import, no problems there.

Along with all this data we've also received a truckload of images, all named like this: `gh001v023.png`, `gh002v458.png` etc. You can see the filename matches the productnumber.

The problem

So usually we would have a Media Picker on the Product node to select the image for it, but that's not an option with 1500 products in the catalog...

So instead of selecting images on all 1500 products, we just need to match up the **productnumber** with the **urlName** of the media.

When bulk uploading images to the media section they all get a `@urlName` according to the filename, without the extension. Pretty darn useful. But the full path to the image will contain a number that we cannot predict or lookup, e.g.: `/media/1234/gh002v458.png`. What to do?

We need a little magic

Fortunately, uComponents has an XSLT extension called `GetMediaByUrlName()` - so this exactly the kind of magic we need.

The Media XML for an image looks like this:

```
<Image id="4736" level="2" nodeName="GH001V023" urlName="gh001v023" nodeTypeAlias="Image">
    <umbracoFile>/media/1234/gh001v023.png</umbracoFile>
    <umbracoWidth>1200</umbracoWidth>
    <umbracoHeight>1200</umbracoHeight>
    <umbracoBytes>1643882</umbracoBytes>
    <umbracoExtension>png</umbracoExtension>
</Image>
```

The Media XML - note that @urlName matches the filename

And here we have the relevant XSLT:

```
<xsl:template match="Product">
    <!-- Lets check if productNumber contains any data,
        otherwise we give at a default value.
        Remember to have an image called 'default' -->
    <xsl:variable name="mediaUrlName">
        <xsl:choose>
            <xsl:when test="productNumber[normalize-space()]">
                <xsl:value-of select="translate(productNumber, $uppercase, $lowercase)" />
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="'default'" />
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

    <!-- Here we use the GetMediaByUrlName from uComponents
        to get the Image selected in the productNumber property.
        And in case it returns multiple items,
        we just select the first [1] -->
    <xsl:variable name="media"
        select="umedia:GetMediaByUrlName($mediaUrlName)[1]" />

    <li>
        <xsl:apply-templates select="productTitle[normalize-space()]" />

    <!-- If the media variable has any content, we apply a template to it,
        and we pass a crop mode to the CropUp extension -->
        <xsl:apply-templates select="$media[normalize-space()]" mode="media">
            <xsl:with-param name="crop" select="'150x170M'" />
        </xsl:apply-templates>
    </li>
</xsl:template>
```

Example of getting a product's image from its product number

I was very happy to not have to instruct someone to pick every single product's image, but yet still be able to use the Media section instead of a flat folder of images.

You can download the complete XSLT here

Merry Christmas everyone!

The importance of great images ... and how you achieve them

— by Niels Steinmeier

Hi everybody. My name is Niels Steinmeier, and I run a small digital company called Vokseværk along with two awesome dudes.

On a daily basis I - one way or another - work with and within Umbraco. As an Editor though, since my programming skills are more or less on a non-existing level - and oh no, not meaning that in any good way ... at all - just ask around! :D

Anyways, since we've got that cleared you might have guessed that you're not going to see any code in this post (*boooo ... we're coders, we demand code!*). Yes yes .. I know, but just bear with me here ... you might even learn something completely different.

Images are my speciality. And images have a huge effect on how a website - or any other digital media for that matter - is presented.

©Niels Steinmeier

We all look to Apple if we want a reference or a benchmark on how to make a great presentation. And YES! Apple have a bunch of great products - I use most of them myself - but if you really look, you'll notice that one thing all Apple presentations have, is GREAT images. Professionally shot, edited and cropped. Flawless.

We should have that too!

Sure we should, and that's the whole reason for this post ... with a darn long intro :D

When prepping a site for a client it's ok to have images of kittens scattered all over the site - for *testing* purposes. But when you get to the point of presenting the site to the client, you should really use great photos that are more representative to the site.

And why should you do this?

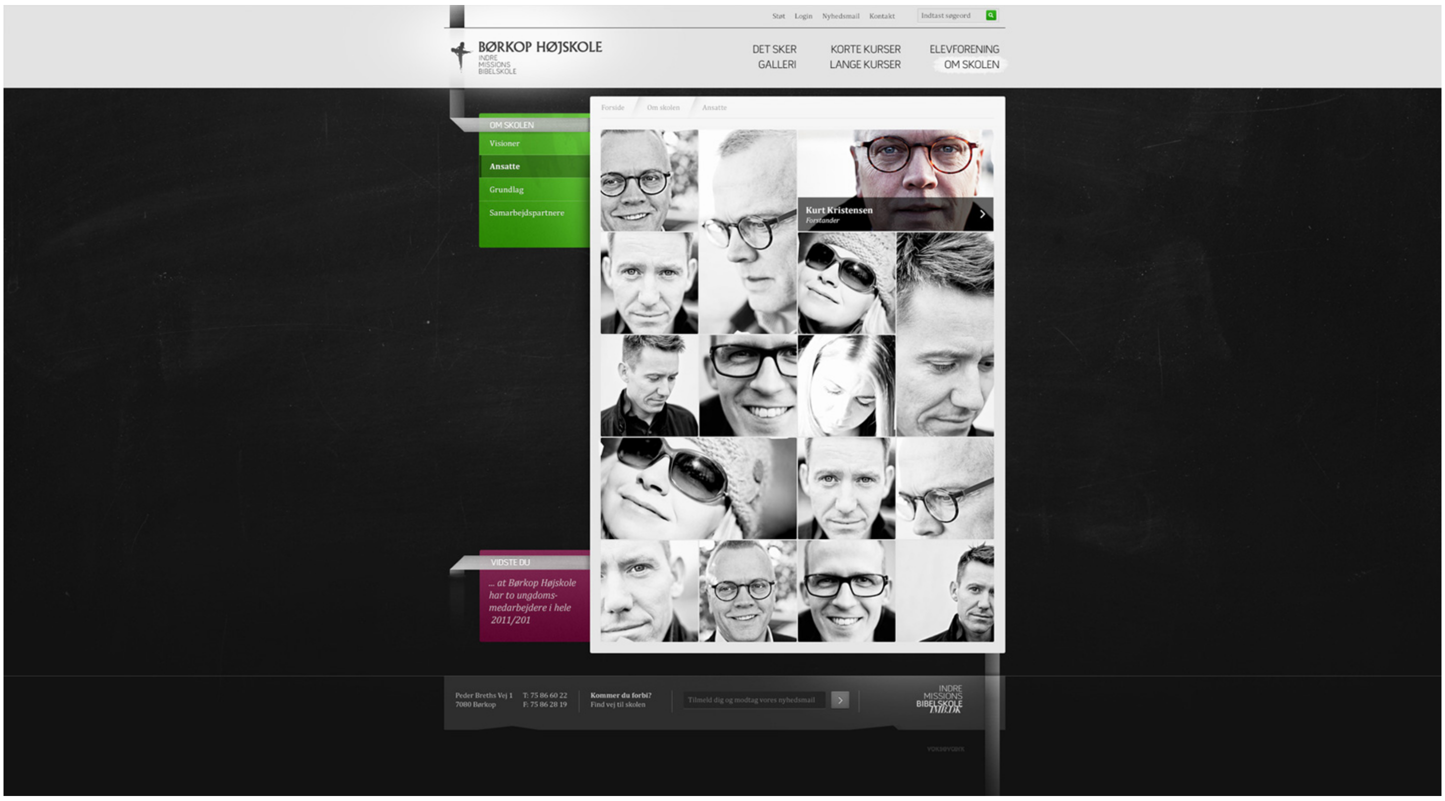
You should because it's my experience, after many years of trial and error, that most clients don't respond positively to "layout" images or "lorem ipsum" texts.

Though you're telling them "*this will all be replaced, of course, with your content*" and they say "*Oh ... I see*". Still, they often don't get it. Furthermore, this is your chance as an angency to guide your clients on the "right path". And by that, of course, I mean YOUR path: Your ideas, your design and maybe even your images.

Let me demo that for you

We had this assignment of totally redesigning a site. It was a school, that - amongst other needs - needed to present the teachers and staff. A quick thing to fail due to often bad images, and/or different angles and crops of the people. So what we did, was using a batch of my various portraits - in different crops - and in black-and-white [*always a good trick, if you have very different images*] showing them, how we envisioned the staff presentation.

Like this *[click for larger view]:*

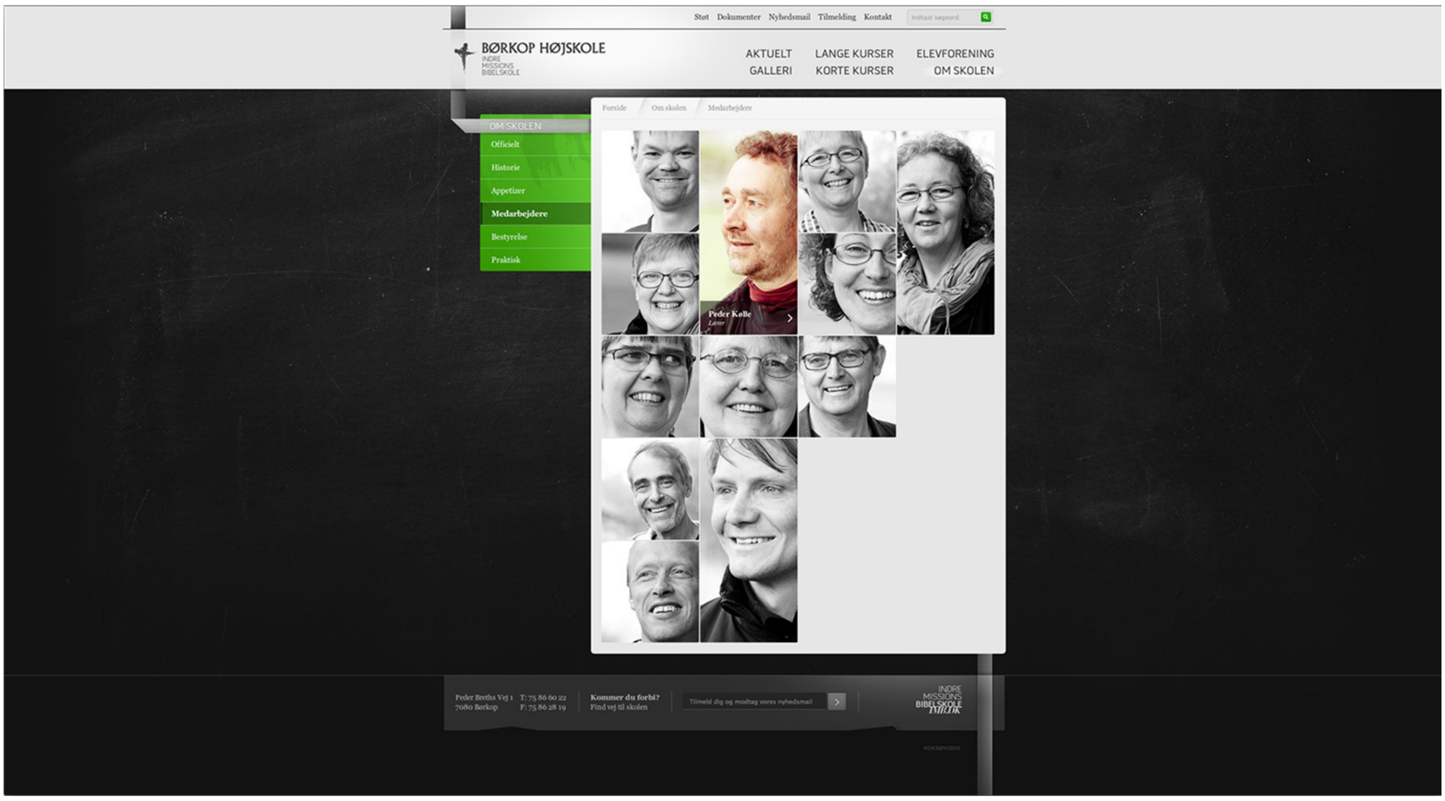

Støt Login Nyhedsmail Kontakt
Indtast søgeord
BØRKOP HØJSKOLE
INDRE MISSIONS BIBELSKOLE
DET SKER
GALLERI
KORTE KURSER
LANGE KURSER
ELEVFORENING
OM SKOLEN
Forside / Om skolen / Ansatte
OM SKOLEN
Visioner
Ansatte
Grundlag
Samarbejdspartnere
Kurt Kristensen
Forstander
VIDSTE DU
... at Børkop Højskole har to ungdoms-medarbejdere i hele 2011/201
Peder Breths Vej 1 7080 Børkop
T: 75 86 60 22 F: 75 86 28 19
Kommer du forbi? Find vej til skolen
Tilmeld dig og modtag vores nyhedsmail

Our Photoshop layout presented to the client

(Just imagine that with kittens instead ... right?)

So what happed was that the client realised, that this would not do with their current - sorry to say - own *[read: bad]* images. So they hired a photographer to shoot and - ta-da - color grade the images, to get that consistent look.

The final result *[click for larger view]:*


Støt Dokumenter Nyhedsmail Tilmelding Kontakt
Indtast søgeord
BØRKOP HØJSKOLE
INDRE MISSIONS BIBELSKOLE
AKTUELT
GALLERI
LANGE KURSER
KORTE KURSER
ELEVFORENING
OM SKOLEN
Forside / Om skolen / Medarbejdere
OM SKOLEN
Officielt
Historie
Appetizer
Medarbejdere
Bestyrelse
Praktisk
Peder Kølle
Lærer
Peder Breths Vej 1 7080 Børkop
T: 75 86 60 22 F: 75 86 28 19
Kommer du forbi? Find vej til skolen
Tilmeld dig og modtag vores nyhedsmail

The final result

One **happy** client ... but also one **happy** agency :)

A little bonus info: We got to do all the crops for the images - and yes, we're using the standard Image Cropper here. The images have no fixed spot in the grid, it all depends on the amount of staff. A new comes in, the layout of the page is rearranged. So each staff image actually have tree crops: One square, one tall and one wide. Umbraco takes care of all that for us - well, actually our mighty coda-Yoda® Greystate wrote some kind of algorithm for that ... but still.

But I don't have any great photos I can use!

Sure you do. Either the client has provided a bunch for you to use, once the design phase begun, or you may have a pile in you system. Otherwise you have have a couple of options:

1. You could use stock-images from i.e. iStock Photo or Shutter Stock
2. Or you could shoot them yourself

If you go with option 1 you should more or less be home free, editing wise. The cool thing about these sites is, that images are fairly cheap. All the way down to $1 pr. image. Beware, some are fairly expensive though. But you can be sure to get quality images.

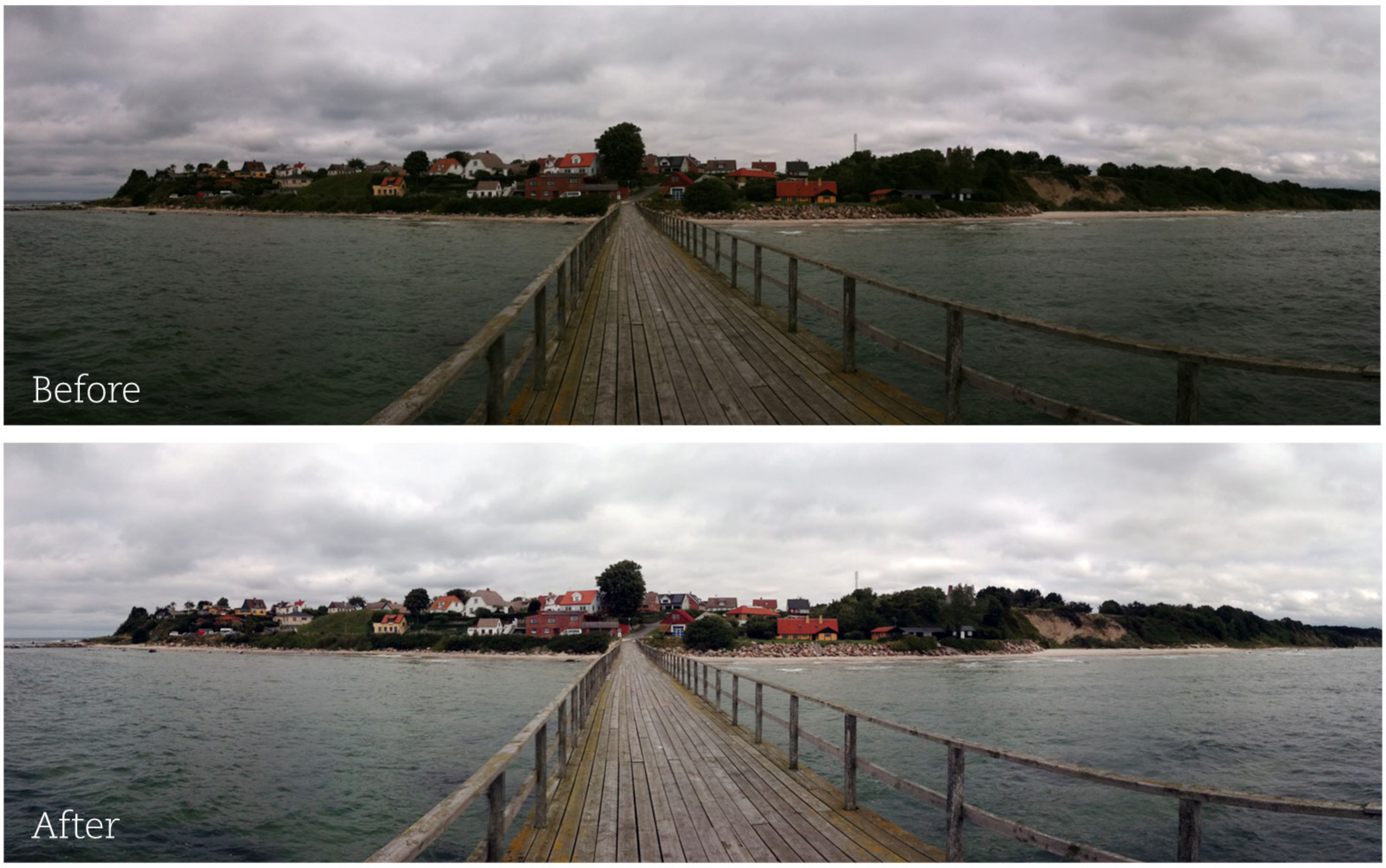
If you go with option 2 and you're not an image editing/color correction expert ... don't worry, I've got you all covered.

All you need is **Adobe Photoshop** or **Adobe Photoshop Elements**. Both apps can be downloaded and tested free for 30 days here.

When you're set, simply load the action I'm providing here on the site [*see bottom of article*] and run it on you images.

What does it do? - **glad** you asked.

Basically it does sort of a "general color correction", adds contrast and sharpening to your images. So let me show you a before and after.

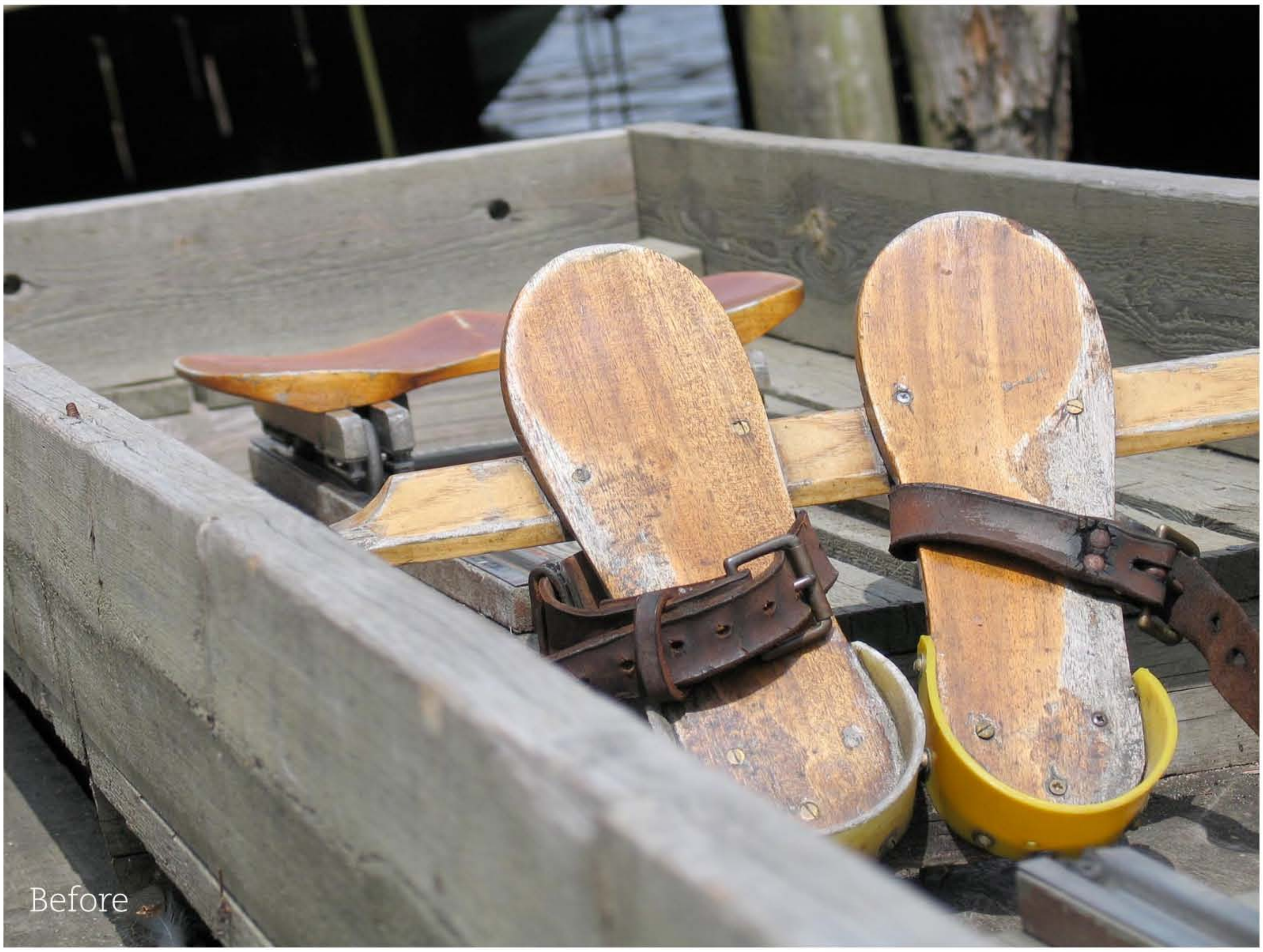


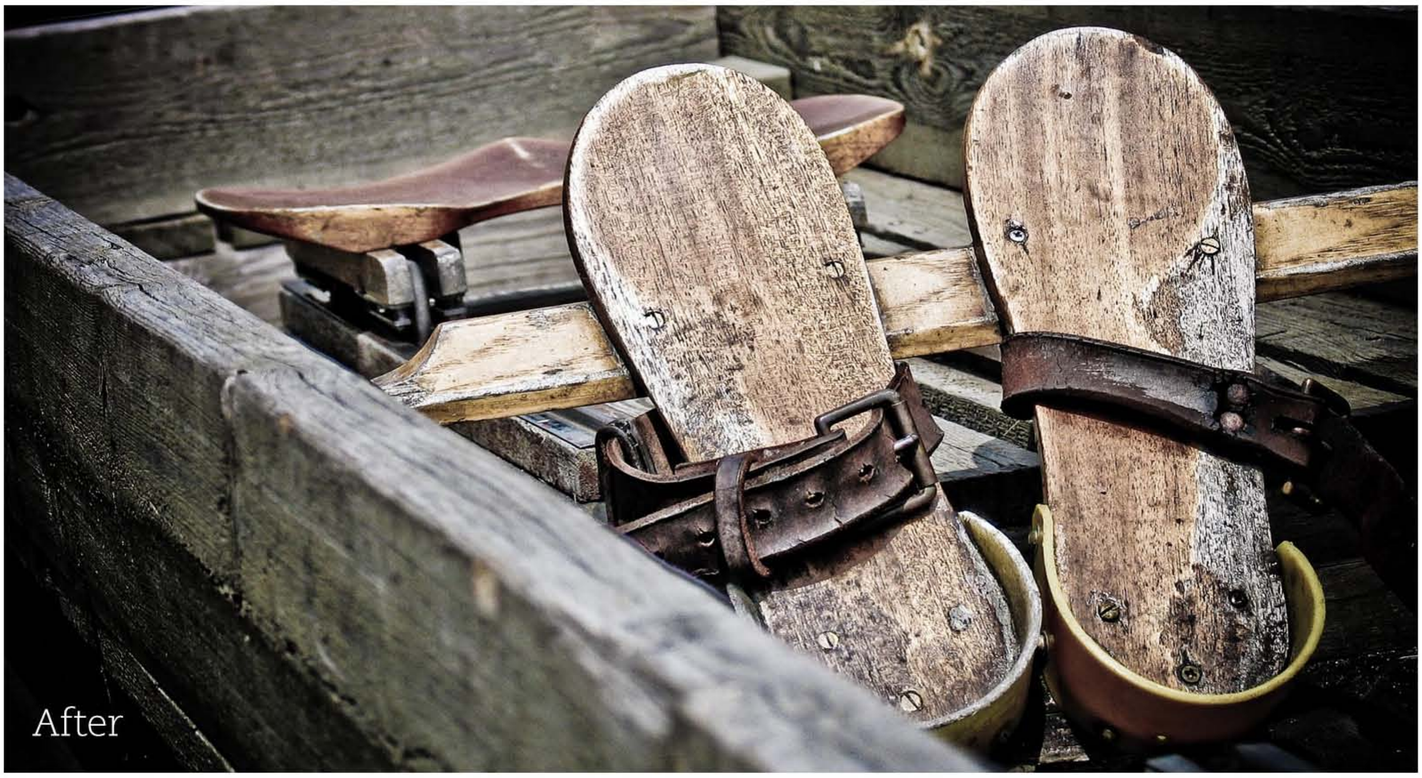
It's not magic, but still the AFTER is MUCH better

The image is more clear now and colors are more vibrant. Believe me, this does make a difference.

And if you *are* a Photoshop expert, then *please* remember to be creative. Play with cropping and color grading. You might just get that extra order from the client.

And as a finishing touch, I'll show you just that.





The result of creative cropping and color grading

I hope you all can see the potential of this ...

And finally ... the Photoshop Action - download it here.

I hope you learned something from this and enjoyed it ... despite the lack of code lines.

Keep those great images flowing ... and have a wonderful Christmas.

And if you're somewhat intrigued to view more of ... let say ... my images, you could check out my website. Thank you for "listening".

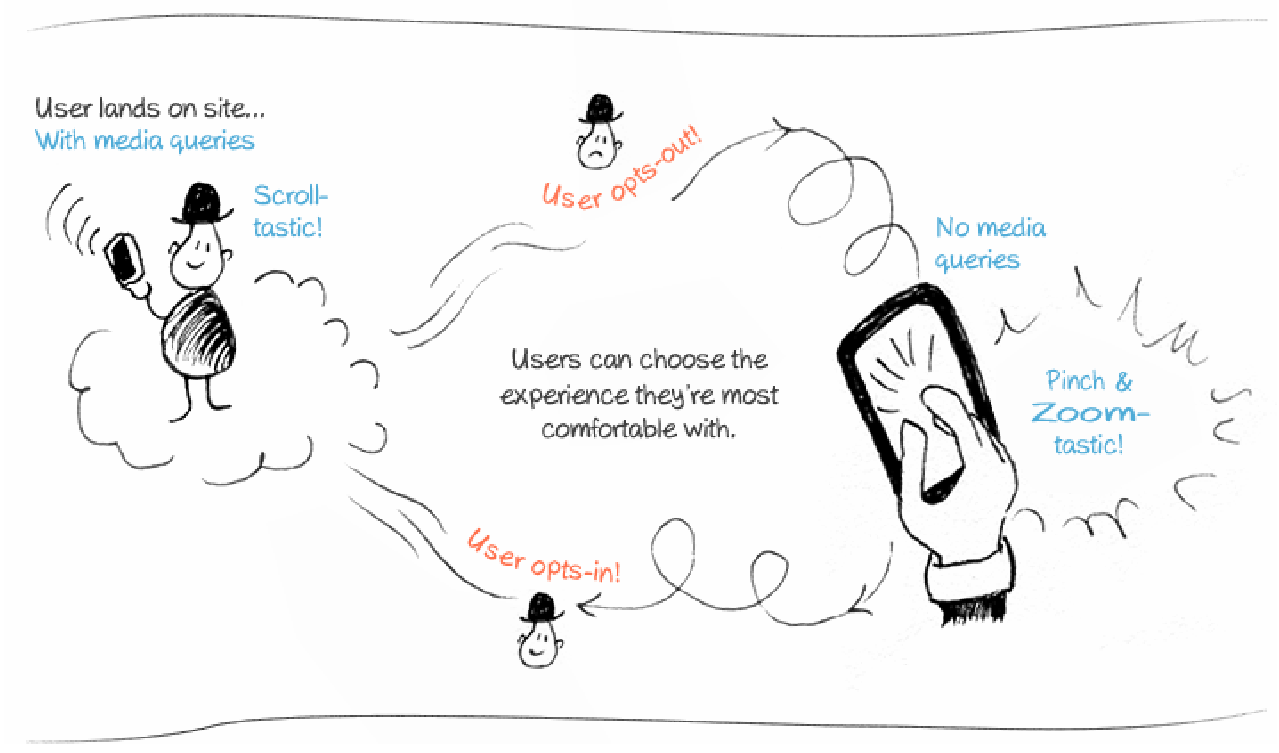
... and to those of you wondering,
YES ... we *are* related ... he is my brother!

Media Query opt-out with Umbraco

— by Tim Clulow

Responsive web design has been around for a good while now, and as nice as media queries are - sometimes you might be on a device that is more than capable of surfing a full website, or perhaps you even prefer to pinch and zoom on a minuscule screen because that's just the way you roll?

Hey it's the web right? - users can choose, we're cool with that.



> tl;dr? See the working example here

Inspired by these posts:
http://mistermorris.tumblr.com/post/19679735499/a-responsive-panacea
http://css-tricks.com/user-opt-out-responsive-design/

I set about seeing if I could piece together a Media Query opt-out with Umbraco

Umbraco lets you be all anal retentive on the front end

Umbraco: the Neve of the CMS world?

As demonstrated beautifully by Dan Okkels Brendstrup, front-enders hold dear the HTML that they write. By design with Umbraco you start from the ground up, so rather than trying to shoehorn your HTML & CSS into an awkward templating system, it works with your way of working from the very start. While this post mostly may not be the most Umbraco specific of the bunch, it's the sheer fact that if you can imagine it, most likely you can do it with Umbraco.

Back to business

For simplicity lets say we have our media queries in a separate style sheet, which will be included in our *all singing, all dancing* version but excluded for our media query opt-out version.
We want to include these styles on a page based on whether the user is happy to receive a media-query based layout or not and set a cookie to remember what their preference is.
Without getting all grown up and creating a macro, lets just include it right within our masterpage where we would usually be including our styles.

Styles and Viewport (Razor)

```
<umbraco:Macro runat="server" language="cshtml">
@{
// check for a previously set cookie, so we know which version to serve.
HttpCookie optOutcss = new HttpCookie("noThanksToMediaQueries");
optOutcss            = Request.Cookies["noThanksToMediaQueries"];

    // Need to check for a null value if there's no cookie
    if(optOutcss != null){

        // fixed viewport and no media queries
        <meta name="viewport" content="width=960, initial-scale=0.3333, user-scalable=yes"/>
        <link rel="stylesheet" href="/css/nomediaqueries.css"/>

    }else{

        // device width viewport with media queries
        <meta name="viewport" content="width=device-width; minimum-scale=0.3; maximum-scale=1;"/>
        <link rel="stylesheet" href="/css/withmediaqueries.css" />
    }
}
</umbraco:Macro>
```

Tip: The same could be done with XSLT. I haven't given myself to the dark side just yet.

Now lets make a separate switch link (which could appear anywhere with our HTML source flow) that the user can click,
perhaps they know the full desktop site very well and want to see it how they would normally.

The switch link (Razor)

```
<umbraco:Macro runat="server" language="cshtml">
@{
// our variable optOutcss with tell us if the user said no to Media Queries
HttpCookie optOutcss = new HttpCookie("noThanksToMediaQueries");
optOutcss = Request.Cookies["noThanksToMediaQueries"];

    // Need to check for a null value if there's no cookie
    if(optOutcss != null){

        // Yep a cookie exist, the user has already opted out of Media Queries,
        // lets give them the option of getting them back in case they're feeling adventurous.
        <div id="mediaqueries-off">Responsive styles are currently off <a href="#">Switch On</a></div>

    }else{

        // The user hasn't set a preference for Media queries yet so they'll get them by default.
        <div id="mediaqueries-on">Responsive styles are currently on <a href="#">Switch off</a></div>
    }
}
</umbraco:Macro>
```

The switch (js)

The final piece of the puzzle. I'm using jQuery & the cookie plug-in here for convenience.
Smarter Umbracians that me would probably also have a non-JavaScript fallback - perhaps by just throwing an `&altTemplate=noMediaQueries` string into the URL and then adjusting the alternate master template accordingly.

```
$(document).ready(function(){

  // When someone opts-out of media queries.
  $("#mediaqueries-on a").click(function(){
    if($.cookie('noThanksToMediaQueries') != "true"){
    //Media queries to the ground!!!
    $.cookie('noThanksToMediaQueries', 'true', { expires: 20, path: '/' });
    location.reload();
    }
  });

  // When someone says gimmie that good Media Query shit.
  $("#mediaqueries-off a").click(function(){
    // Remove the cookie, The full site wasn't all that.
    // Get me back to the mini version.
    $.removeCookie('noThanksToMediaQueries', { path: '/' });
    location.reload();
    return false;
  });

});
```

You'd probably want to also include a non-js fallback in a real world scenario

And there you have it - See the example here

Some web designers may be loathed to see all their hard work creating a responsive design undone, but the way I see it most users are going to be either appreciative or completely oblivious that they're being served a responsive design, and this is just one way of helping out users who know exactly what they want and dont want to be spoon fed, or feel like they're being hand-held just because they're on a certain screen-width or device. The particularly cranky or ultra savvy users may even be circumnavigating your design altogether with services like Instapaper and Readability anyway. So ease up! Design still matters.

Wishing you a very Kentucky Fried Christmas from Japan

Online Video Clip

Who Picked This ?

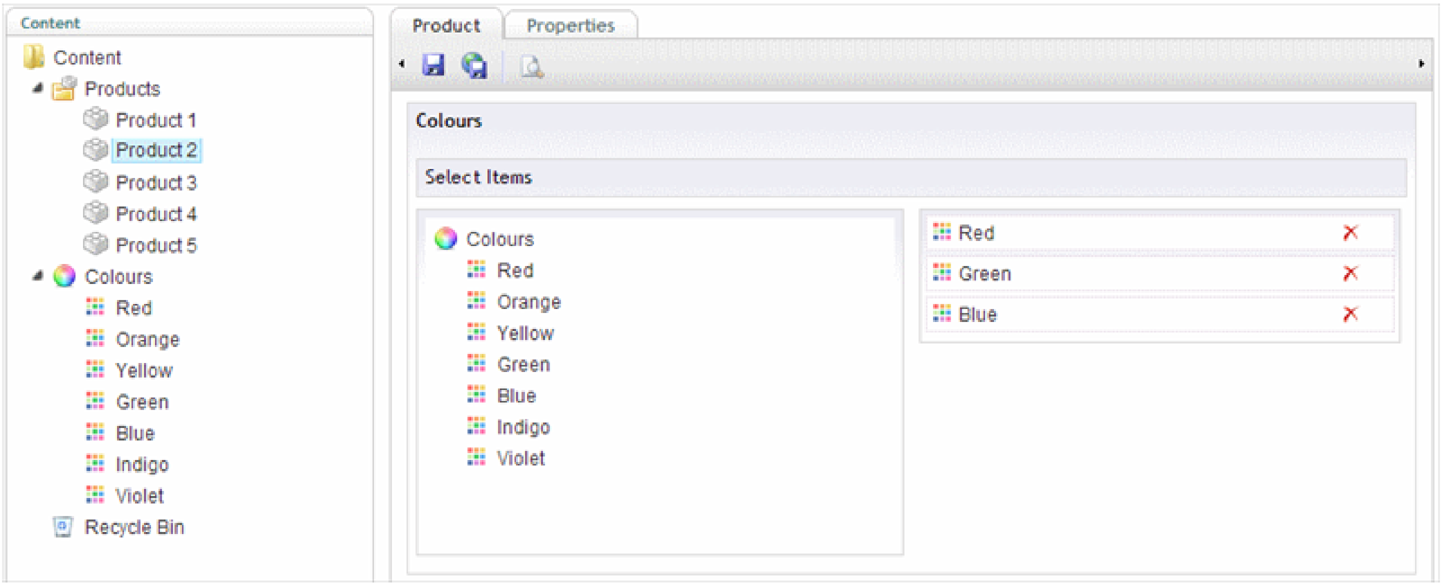
— by Hendy Racher

Here's a feature that can be added to sites using pickers* to show on a picked node (be that a content, media or member item) any other nodes that may have selected it - it uses two DataTypes: **Relation Links** from uComponents and **Picker Relations** (Umbraco 4.9+) where both are wired-up by a shared Relation Type. (No code required).

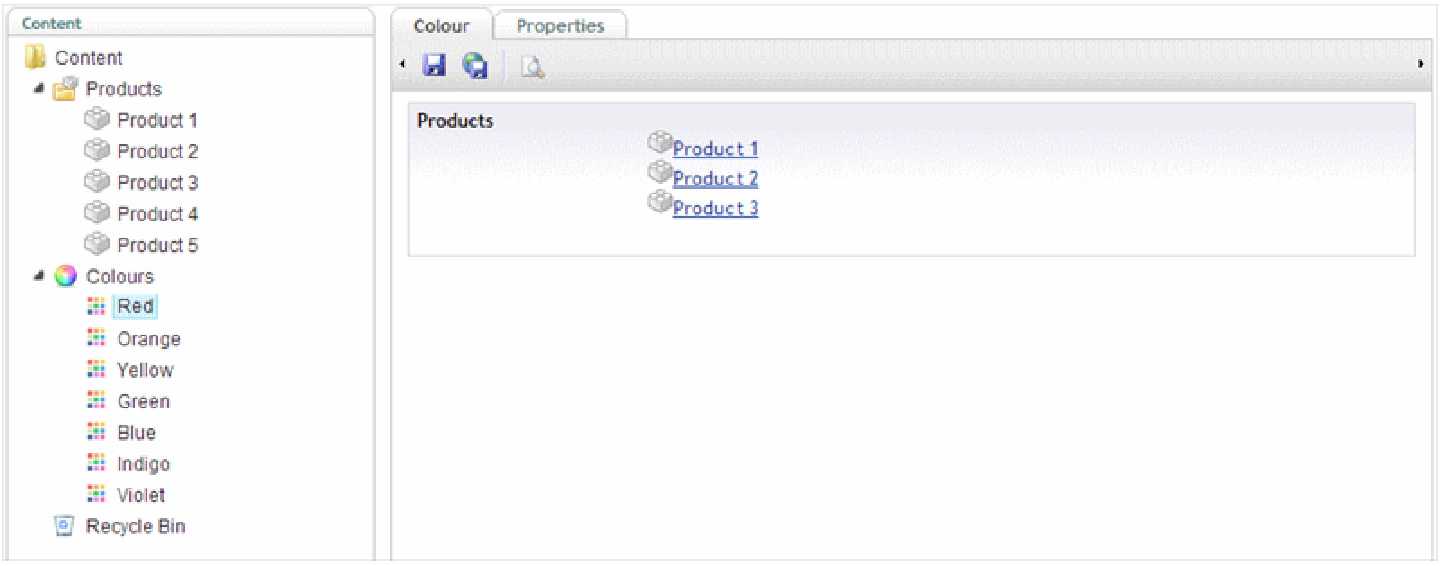
* Compatible pickers include any that store either a single Id, a CSV of Ids or known XML fragments.

These backlinks can be really useful to content editors, as it's a quick way to see and navigate around related items. For example, on an image, see a list of links to all pages where it's been picked as a heading or for use in a carousel...

To demonstrate, we'll create a content structure where products can pick colours (via a Multi-Node Tree Picker):



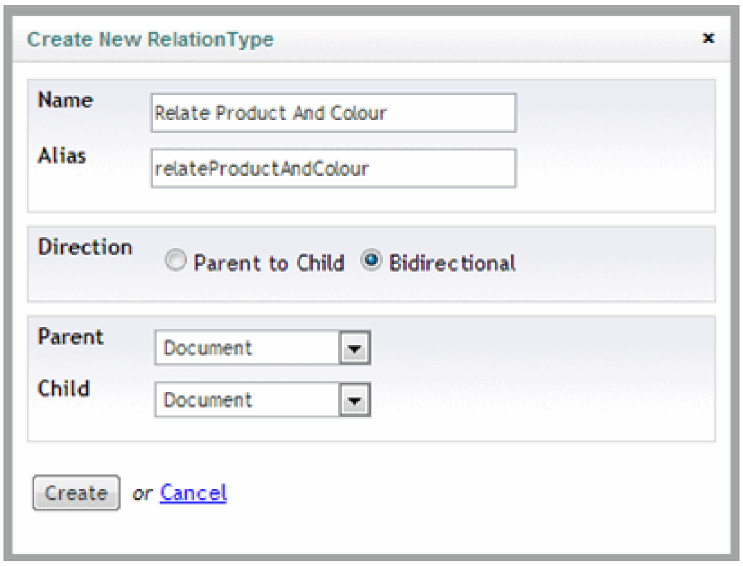
with a view to rendering a list of links back to the related products on each of the colour nodes:



Products: Product 1, Product 2, Product 3

1) Create Relation Type

First we'll create a RelationType. This'll be used to store the associations between products and colours.



Name: Relate Product And Colour
Alias: relateProductAndColour
Direction: ( ) Parent to Child [✓ Bidirectional]
Parent: Document
Child: Document

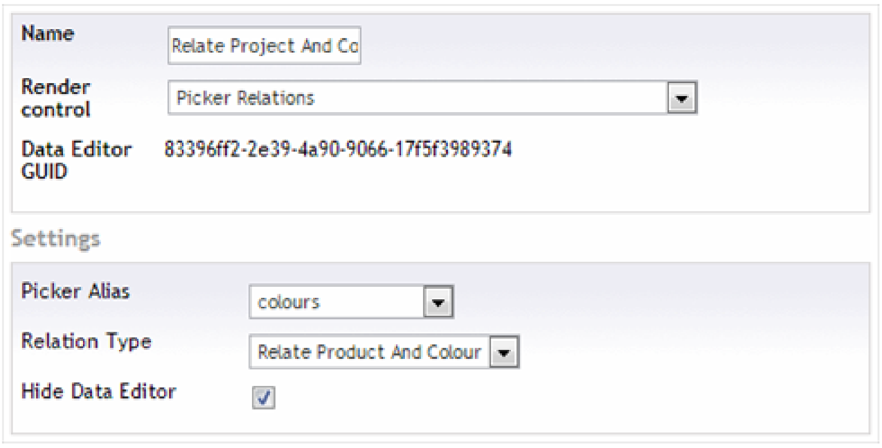
The naming convention using the word 'To' for Parent to Child types, and the word 'And' for Bidirectional types seems to work quite well - for example, relateProductAndColour indicates that both the Parent and Child fields will be checked, so it makes no difference which field the colour and product Ids are in (whilst if it was named relateProductToColour, it would imply a direction where lookups are performed on the Parent Product Id only).

The Direction has been set to Bidirectional, as we just want to know if two Ids are related - using a Parent to Child type could prove beneficial if the direction was significant (or for micro performance tuning).

Parent and Child types have been set to use Documents, as products and colours are both items in the content tree.

2) Create Picker Relations

Next we'll create a DataType using Picker Relations - this will watch a picker and automatically create and/or delete relations such that they remain in-sync with the items selected by the picker, in this case the colours selected by a product.



Name	Relate Project And Co
Render control	Picker Relations
Data Editor GUID	83396ff2-2e39-4a90-9066-17f5f3989374

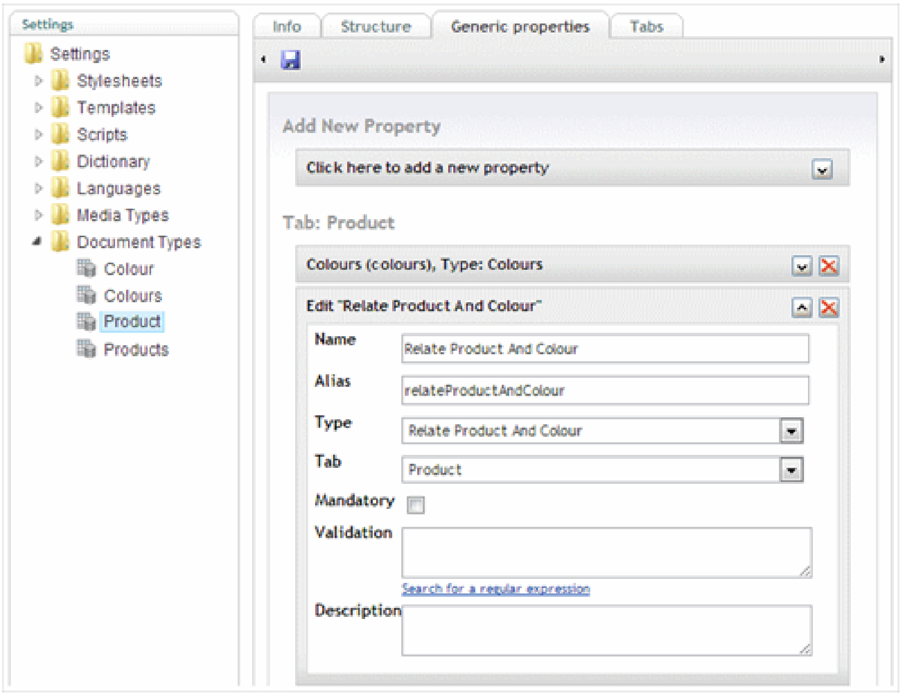
Settings

Picker Alias	colours
Relation Type	Relate Product And Colour
Hide Data Editor	✓

The value for the Picker Alias (as above) will be the *Alias* of the **Colours** MNTP property on the Product DocumentType (see screenshot below), the Relation Type refers to the one just created, whilst checking Hide Data Editor prevents this DataType from being visible as no UI required (it can however be useful to uncheck this option if debugging, to confirm that the Picker Alias was found on the current item).

3) Add Picker Relations to Product

The DataType can be added on any tab, or at any level in the hierarchy (so long as at render time a matching Picker Alias can be found) - usually place it next to the Picker it's watching, so it's easy to identify.



Tab: Product

Colours (colours), Type: Colours

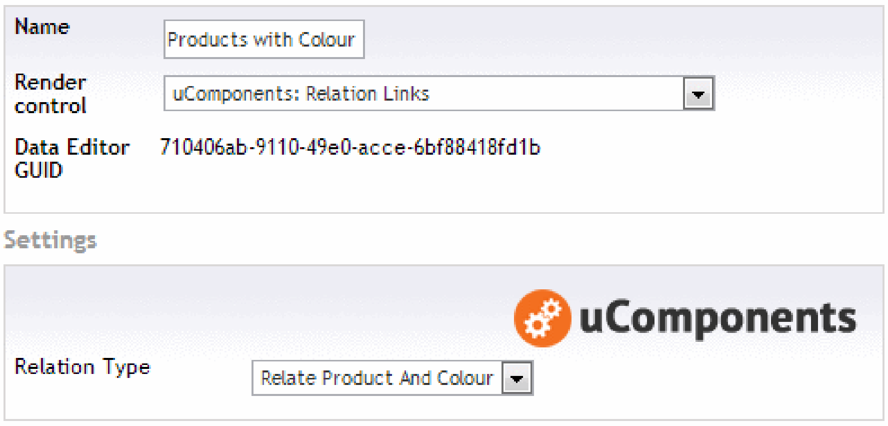
Edit "Relate Product And Colour"

Name	Relate Product And Colour
Alias	relateProductAndColour
Type	Relate Product And Colour
Tab	Product
Mandatory	☐
Validation	
Description	

4) Create Relation Links

The second DataType to create uses Relation Links. Configuration is simple, it just needs a single

RelationType and in this example we're using the the one created in the first step.



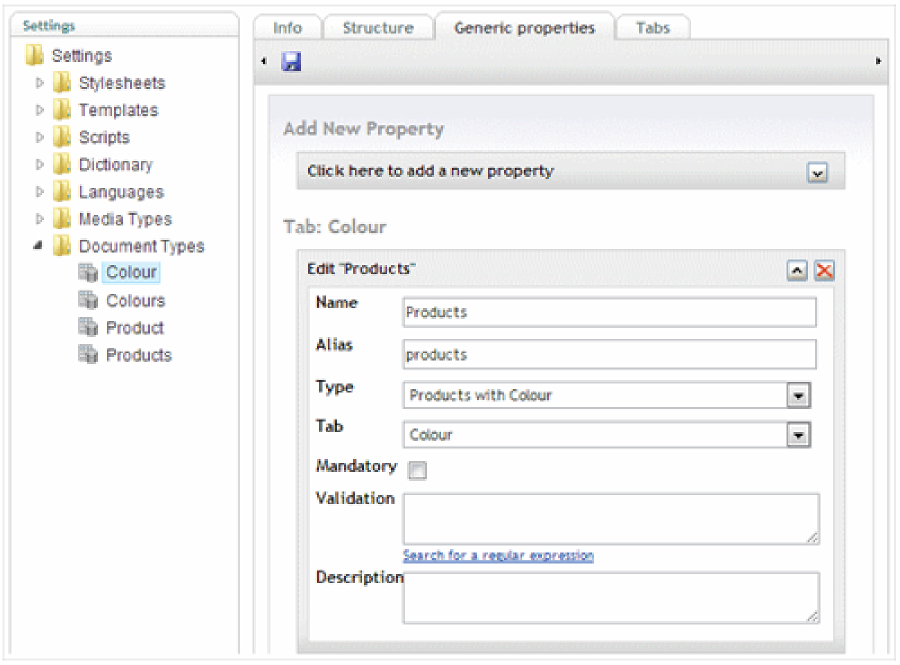
Name	Products with Colour
Render control	uComponents: Relation Links
Data Editor GUID	710406ab-9110-49e0-acce-6bf88418fd1b

Settings

Relation Type	Relate Product And Colour

5) Add Relation Links to Colour

Once added to the Colour DocumentType, it will query the relations data and render a list of links back to any related Products - it simply queries the configured RelationType for anything that relates to the Id of the item currently being viewed / edited.



Name	Products
Alias	products
Type	Products with Colour
Tab	Colour
Mandatory	☐
Validation	
Description	

Razor and Faceted Search In Umbraco

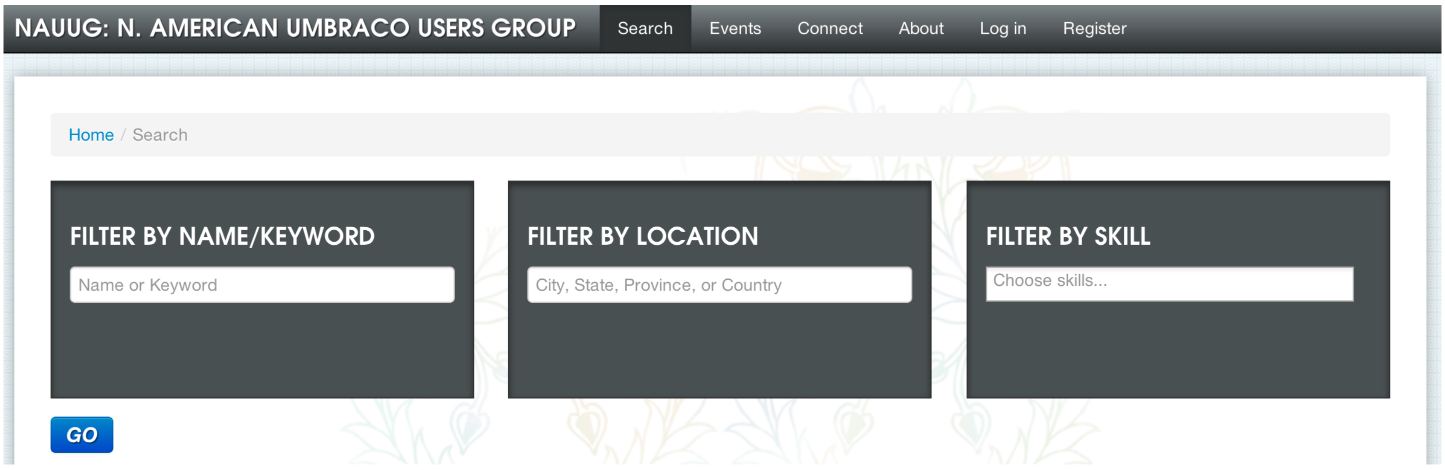
— by Jason Prothero

A little over a year ago we had a few sites that needed faceted search of content nodes for a specific Document Type. At the time Razor had just been added to the core and I was finding excuses to create macros in Razor. Rather than implementing the faceted search in XSLT or straight .NET User Controls, I went straight to Razor. However, I found using the Umbraco Razor API challenging for search so I built my own workaround that has worked well for us and our clients.

In doing so I found some solutions to two key problems:

- Multiple selection facets to multiple selection properties in Razor
- Complex DynamicNode sorting using Razor

I also used this technique to help quickly add a somewhat complex filter to the North American Umbraco Users Group site. This will be my example for this post.



(Full disclosure: I'm not necessarily proposing this as a "best practice", but hope it will either help a few people or simply spark a conversation about better ways to get the same outcome (which I welcome). Also, this was before Niels Kühnel's awesome Faceted Search with Examine demo at Codegarden '12 which I still want to figure out how to do.)

Problem #1: Multiple selection facets to multiple selection properties in Razor

Currently each profile at the NAUUG site has a set of skills that they can select. In the Umbraco admin this is simply content in a folder that is selected with a Multi-Node Tree Picker. On the Search page, there is an advanced search that allows for search by skills. Multiple skills can be selected. If none are selected, the assumption is that all profiles will be shown that match the other filters.

This was a difficult problem to solve for me using the the standard `.Where()` method in the Umbraco Razor API. After trying my best, I gave up and simply looped through the DynamicNodeList and did my own conditional check.

This is where I ran into my first problem. I couldn't change the `DynamicNodeList` to remove the items I didn't want. I also couldn't add to an empty list. But I could just create a `List<DynamicNode>` and add to that. Sweet!

```
// Get your set of nodes to loop through (could be any query including Where())
var personListings = @Model.NodeById(parent).Descendants("Person");

// Place to store matched profiles
List<DynamicNode> possibleListings = new List<DynamicNode>();

// Loop through and add
foreach(dynamic item in profileListings)
{
    bool includeFromSkills = true;
    // **** Filter code goes here

    // Include in results if matches
    if( … // **** other checks
            && includeFromSkills )
    {
      possibleListings.Add(item);
    }
}
```

Now we need to actually filter the nodes. I borrowed a technique I saw first in the XSLTSearch package from Douglas Robar to add commas around a comma separated list and the search term and do a Contains check on a string. This means my Multi-Node Tree Picker needed to be set to CSV instead of XML and the list of filtered skills needed to be a list of node ids (comma separated).

I've now added the param passed in, added of the commas to the filter (outside of the foreach), and the skills filtering code.

```
// Get params
var skills = String.IsNullOrEmpty(Parameter.Skills) ? "all" : Parameter.Skills;

// Get your set of nodes to loop through (could technically be any query)
var personListings = @Model.NodeById(parent).Descendants("Person");

// Place to store matched profiles
List<DynamicNode> possibleListings = new List<DynamicNode>();

// Ids of skills that the User has selected to filter results with
string skillsListJoined = String.Format(",{0},", skills);

// Loop through and add
foreach(dynamic item in profileListings)
{
    /////////////////////////////////////////
    // Skills search
    bool includeFromSkills = true;
    if( skills != "all" )    {
      includeFromSkills = false;
      string profileSkills = item.GetProperty("skills").Value;
      if( profileSkills.Length > 0 )
      {
        foreach(var skill in profileSkills.Split(','))
        {
          if( skillsListJoined.Contains(String.Format(",{0},", skill)) )
          {
            includeFromSkills = true;
            break;
          }
        }
      }
    }
    /////////////////////////////////////////

    // Include in results if matches
    if( … // other checks
            && includeFromSkills )
    {
      possibleListings.Add(item);
    }
}
```

The skill filtering does do a little more processing than I would like (the second foreach loop), but it hasn't affected performance considerably from the implementations we have done so far. I would recommend doing only the things you absolutely have to in the loop. For example, don't do extra node lookups or searches that can be done outside of the top level foreach.

Problem #2: Complex DynamicNode sorting using Razor

Now imagine you want to sort by some properties on each node like "levels" or "type" with a fallback to alphabetical when the properties are equal.

In this case we're sorting by type (business/person) then alphabetical. On other projects we have built there have been member levels and then even "auction" values that allow some listings to float up based on a paid amount (like Google AdWords). Basically, you get full control over the sort.

The nice thing is that by using a standard .NET `List<>` object we have access to the `Sort()` method and can provide our own delegate to handle the sorting logic. The unfortunate thing we found is that you really have to watch what you put in the delegate or things can slow down quickly. That is why there are ids in the sort logic. The types are ids in the tree selected via Ultimate Picker. I found that when we did a node lookup based on strings or names, it slowed things down considerably.

Here is the delegate code I used:

```
    possibleListings.Sort(delegate(DynamicNode x, DynamicNode y)    {
      dynamic xListingLevel = x.GetProperty("listingType").Value;
      dynamic yListingLevel = y.GetProperty("listingType").Value;

      int xSort = 2;      // default
      if( xListingLevel == "3245" ) { xSort = 1; }  // listing with more importance (direct id for
speed)

      int ySort = 2;
      if( yListingLevel == "3245" ) { ySort = 1; }

      if( xSort == ySort )
      {
        return x.Name.CompareTo(y.Name);
      }
      else
      {
        return xSort.CompareTo(ySort);
      }
    });
```

Hope that helps someone out there! Or perhaps you have a different way to solve these problems?

Oh, and if you're located in North America join the North American Umbraco Users Group and tell your friends to do the same! We need to unearth all the Umbraco goodness that's over here and hope to have a North America Umbraco Festival someday! Join the Roll Call!

Happy Holidays!

Creating a Login form with Umbraco MVC SurfaceController

— by Jonas Eriksson

I really like the new support for MVC in Umbraco. It's certainly great to be able to use the standard Razor Layout system instead of old Masterpages. And together with the new strongly typed API, I think it's now quicker and nicer than ever to build the front end of an Umbraco site.

Some time ago I built a RazorLogin package for Umbraco, using "WebPages style" = "throw logic and view into the same file". It does still work in Umbraco 4.10. But, as Mr. Sebastiaan Janssen rightfully commented in a twitter dialogue about RazorLogin:

> **"Would recommend a REAL (Surface)Controller in 4.11 though. Views shouldn't really have logic."**

Ok, I don't want to promote bad habits (well, not too often anyway), so here we go:

Note that some of the code is abbreviated to keep the article a bit shorter, see the Gist link below for the complete source code.

The Model

In MVC we use a Model to describe our data, a model is simply a class. For the login form we need strings for username and password and a bool for a "remember me" checkbox.

```
public class MemberLoginModel
{
    public string Username { get; set; }
    public string Password { get; set; }
    public bool RememberMe { get; set; }
}
```

The Controller

The C in MVC is as we know the Controller. It deals with user actions which returns a result, normally a view or a redirection. In our case we write one function to handle the *initial get request*, one to handle *logout* and one to handle a *login form post* from the user.

To make it work as a surface controller we simply inherit the class from `SurfaceController`:

```
public class MemberLoginSurfaceController : Umbraco.Web.Mvc.SurfaceController
{

    // The MemberLogin Action returns the view, which we will create later. It also instantiates a
new, empty model for our view:

    [HttpGet]
    [ActionName("MemberLogin")]
    public ActionResult MemberLoginGet()
    {
        return PartialView("MemberLogin", new MemberLoginModel());
    }

    // The MemberLogout Action signs out the user and redirects to the site home page:

    [HttpGet]
    public ActionResult MemberLogout()
    {
        Session.Clear();
        FormsAuthentication.SignOut();
        return Redirect("/");
    }

    // The MemberLoginPost Action checks the entered credentials using the standard Asp Net
membership provider and redirects the user to the same page. Either as logged in, or with a message
set in the TempData dictionary:

    [HttpPost]
    [ActionName("MemberLogin")]
    public ActionResult MemberLoginPost(MemberLoginModel model)
    {
        if (Membership.ValidateUser(model.Username, model.Password))
        {
            FormsAuthentication.SetAuthCookie(model.Username, model.RememberMe);
            return RedirectToCurrentUmbracoPage();

        }
        else
        {
            TempData["Status"] = "Invalid username or password";
            return RedirectToCurrentUmbracoPage();
        }
    }
}
```

TempData is an UmbracoMVC thing that can be used to add data to a view without adding it to the view model. In standard MVC we're used to ViewData (or the dynamic ViewBag). *(In standard MVC we would probably return the ModelState and the model* *to the view on failed login with a message. But I could not get that to work in Umbraco currently. See also below. TempData works nice though.)*

The View

The last letter in our acronym is V for View. It is a Razor file, and as opposed to our class files, it is saved directly in the web site. In the /Views/Partials folder with the name MemberLogin.cshtml:

```
@model UmbracoLogin.MemberLoginModel
@if (User.Identity.IsAuthenticated)
{
    <p>Logged in: @User.Identity.Name</p>
    <p>@Html.ActionLink("Log out", "MemberLogout", "MemberLoginSurface")</p>
}
else
{
    using (Html.BeginUmbracoForm("MemberLogin", "MemberLoginSurface"))
    {
    @Html.EditorFor(x => Model)
    <input type="submit" />
    }

    <p>@TempData["Status"]</p>
}
```

By defining a strongly typed model we get nice intellisense when we create our view.

We're having the form created for us automatically with the `BeginUmbracoForm` method together with the `EditorFor` MVC method. The latter adds a label and a suitable input field for each property on the model.

Placing the login form in an actual template

The final piece to the puzzle is to render our HTML in the right place in a suitable template. To do that, just add this line of code:

```
@Html.Action("MemberLogin","MemberLoginSurface")
```

And there you go - a working login form using MVC with SurfaceController!

Spice it up with DataAnnotations

It's really easy to add custom label texts, and validation just by adding some attributes to our model. Let's check that out. First add a reference to `System.ComponentModel.DataAnnotations`. Then add some data annotations to the model:

```
public class MemberLoginModel
{
    [Required, Display(Name = "Enter your user name")]
    public string Username { get; set; }

    [Required, Display(Name = "Password"), DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me")]
    public bool RememberMe { get; set; }

}
```

With these in place the password box will be rendered as an HTML password input, and the labels will be a bit nicer than simply the propertynames.

Add some validation

The `DataAnnotations` attributes also adds validation rules, such as "Required". To let MVC check those for us we need to add one line in our controller:

```
if (ModelState.IsValid)
{
    if (...
```

before the actual password validation. Unfortunately, like I mentioned, I could not make the ModelState work with the SurfaceController. Otherwise we would get validation messages back to the form automatically.

The good news is that client side validation works as expected by using default MVC unobtrusive validation (with jQuery validate).

Automatic client side validation

Add three JavaScript libraries to your layout:

```
<script src="/scripts/jquery-1.8.2.js"></script>
<script src="/scripts/jquery.validate.js"></script>
<script src="/scripts/jquery.validate.unobtrusive.js"></script>
```

Then enable client side validation just by adding these two lines in your web.config `<appSettings>`:

```
<add key="ClientValidationEnabled" value="true"/>
<add key="UnobtrusiveJavaScriptEnabled" value="true"/>
```

Thats it! With these in place the user will get nice messages if he/she forgot to enter text in either of the fields.

Customize your form

If you like to define your form manually it is also easily possible - just the standard MVC behaviour. You can use either or combine:

- *write the raw HTML (use the same names as the model properties):*

```
<input type="text" name="Username"/>
```

- *define each tag individually using Html helpers:*

```
@Html.LabelFor(model => model.UserName, "Friendly Name")
@Html.TextBoxFor(model => model.UserName, new { style = "width: 500px;" })
```

- *create custom EditorTemplates (see references below).*

Happy holidays everyone!

The full source for the login control :

https://gist.github.com/4336141

References and further reeding :

About MVC in Umbraco 4.10+

http://umbraco.com/follow-us/blog-archive/2012/10/30/getting-started-with-mvc-in-umbraco-410.aspx

http://our.umbraco.org/documentation/Reference/Mvc/forms

http://our.umbraco.org/forum/developers/api-questions/36614-411-Using-SurfaceController-Child-Action-with-Post

About DataAnnotations

http://weblogs.asp.net/srkirkland/archive/2011/02/23/introducing-data-annotationsextensions.aspx

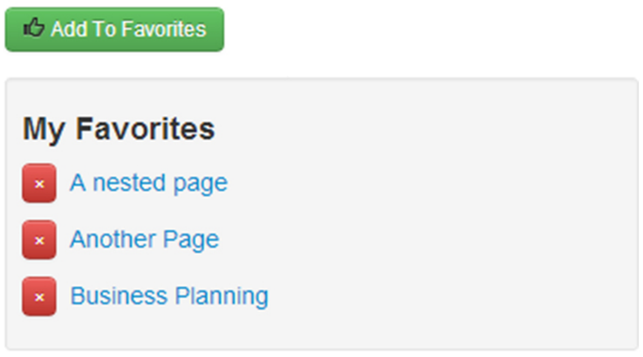
Custom EditorTemplates in MVC

http://coding-in.net/asp-net-mvc-3-how-to-use-editortemplates/

Favorite Pages for Members

— by Bob Baty-Barr

So, you have your Umbraco site all wired up for members, they can log in, adjust their profiles, etc. but wouldn't it be great if they could save a list of page favorites with their profile and easily navigate your site when they are logged in?



Front end display

There was a recent Umbraco blog post about doing something similar with Umbraco relations - check it out here - which involves using the API and some .Net coding.

Wouldn't it be even more fun if you did not have to launch Visual Studio to make this all work?

Below is a brief description of how I achieved just that with the help of some great packages and little bit of razor script and jquery.

The tools I used...

- uComponents Multi-Node Tree Picker
- Macro Service

If you are not familiar with Macro Service, you should check it out!

Setting up the member...

In my membership configuration, I created a dataType for myFavorites of type Multi-Node Tree Picker and added it to the memberType with an alias of myFavs.

The razor macros... Get to it already!

addFavorite.cshtml

```
@using umbraco.cms.businesslogic.member;
@using umbraco.NodeFactory;
```

```
@{
  var member = Member.GetCurrentMember();
  String myValue = @Model.Id.ToString();
  List<string> ListPages = null;

    if (member != null)
        {
            if (member.getProperty("myFavs")!= null && member.getProperty("myFavs").Value != null)
                {
                   string value = member.getProperty("myFavs").Value.ToString();
                 if(String.IsNullOrEmpty(value)){
                           member.getProperty("myFavs").Value = myValue;
                                  } else {
                                        ListPages = value.Split(',').ToList();
                                                 if (!ListPages.Contains(myValue))
                                                    {
                                                     ListPages.Add(myValue);
                                                     member.getProperty("myFavs").Value = String.Join(",",
ListPages.ToArray());

                                                     }
                                           }
                  member.Save();
                  value = member.getProperty("myFavs").Value.ToString();
                  ListPages = value.Split(',').ToList();
                }
            }

}

<ul class="unstyled">

@foreach (var id in ListPages) {
                         var name = umbraco.library.GetItem(Int32.Parse(id),"nodeName");
                         <li><a href="#" rel="@id" class="btn btn-mini btn-danger delete">&times;</a>&nbsp;&nbsp;<a
href="@umbraco.library.NiceUrl(Int32.Parse(id))">@name</a></li>
                                }
</ul>
```

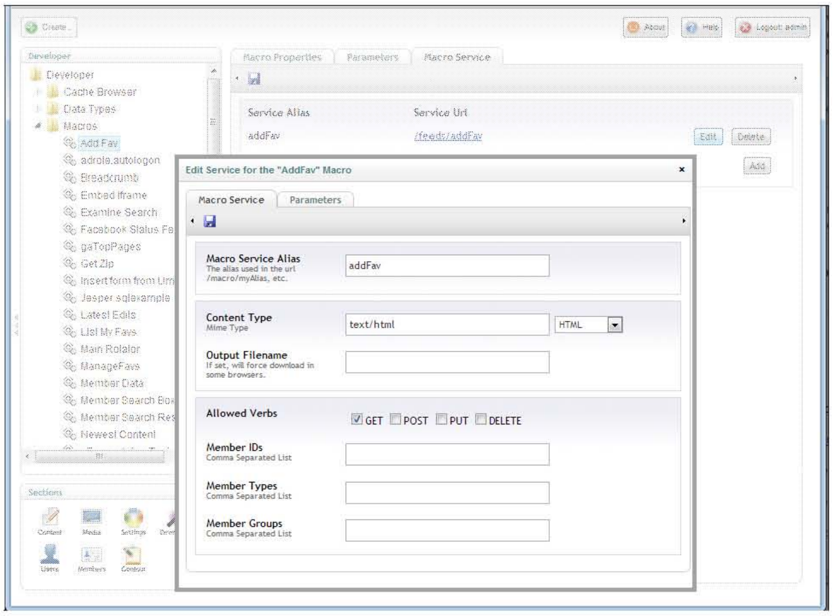
Add Favorite

removeFavorite.cshtml

```
@using umbraco.cms.businesslogic.member;
@{

        var member = Member.GetCurrentMember();
        String myValue = @Model.Id.ToString();
        List<string> ListPages = null;

            if (member != null)
            {
                if (member.getProperty("myFavs")!= null && member.getProperty("myFavs").Value != null)
                {

                    string value = member.getProperty("myFavs").Value.ToString();
                    ListPages = value.Split(',').ToList();

                    if (ListPages.Contains(myValue))
                    {
                        ListPages.Remove(myValue);
                        member.getProperty("myFavs").Value = String.Join(",", ListPages.ToArray());
                        member.Save();
                    }
                }
            }
}
<ul class="unstyled">
@foreach (var id in ListPages) {
                         var name = umbraco.library.GetItem(Int32.Parse(id),"nodeName");
                         <li><a href="#" rel="@id" class="btn btn-mini btn-danger delete">&times;</a>&nbsp;&nbsp;<a
href="@umbraco.library.NiceUrl(Int32.Parse(id))">@name</a></li>
                                }
</ul>
```

Setting up macro service...

So, now that you have the razor macros created, you need to set up the macro service(s). When you install the Macro Service package, it adds a tab to your macros. Simply select the macro you want to turn into a service and set up as you see below. For more info about macro service, please consult the package page on our.umbraco.org.



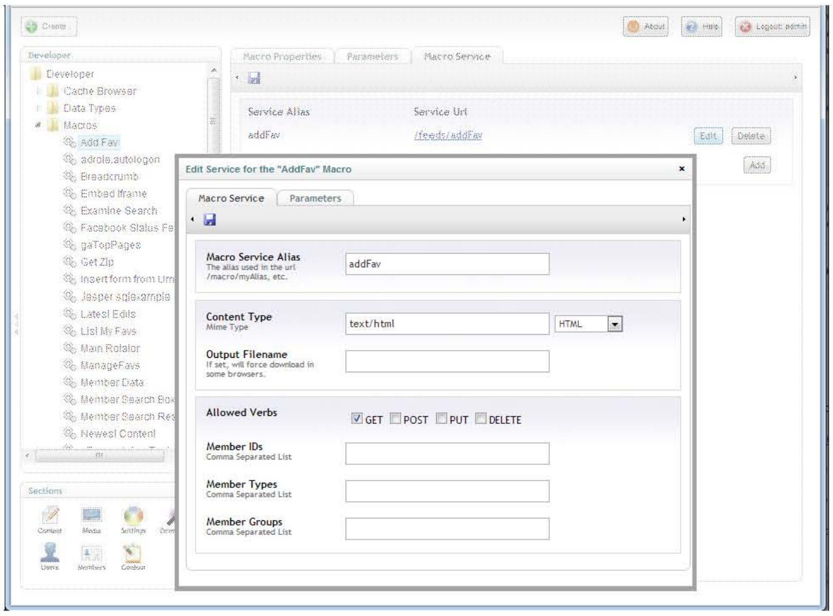
Service Alias	Service Url
addFav	/feeds/addFav

Edit Service for the "AddFav" Macro

Macro Service Alias The alias used in the url /macro/myAlias, etc.	addFav
Content Type Mime Type	text/html HTML
Output Filename If set, will force download in some browsers.	
Allowed Verbs	[✓ GET] POST PUT DELETE
Member IDs Comma Separated List	
Member Types Comma Separated List	
Member Groups Comma Separated List	

Configure Macro Service

and...



Service Alias	Service Url
addFav	/feeds/addFav

Edit Service for the "AddFav" Macro

Macro Service Alias The alias used in the url /macro/myAlias, etc.	addFav
Content Type Mime Type	text/html HTML
Output Filename If set, will force download in some browsers.	
Allowed Verbs	[✓ GET] POST PUT DELETE
Member IDs Comma Separated List	
Member Types Comma Separated List	
Member Groups Comma Separated List	

Parameters Tab

Add the button to the page...

We use the `rel` attribute on the link to tell our macro what page to add to the member's favorites.

```
<a class="btn btn-small btn-success" href="#" id="addFav" rel="<umbraco:Item field="pageID" runat="server" />" ><i
class="icon-thumbs-up"></i> Add To Favorites</a>
```

Some jQuery to tie it all together...

Use this jQuery snippet to add pages to the favorites box -

```
$('a#addFav').live("click", function(){
      var theNode = $(this).attr('rel');
      $('#favList').load('/feeds/addFav?nodeId='+ theNode);
      })
```

jQuery snippet

and remove a favorite with this little bit -

```
$('a.delete').live("click", function(){
      var theNode = $(this).attr('rel');
      $('#favList').load('/feeds/RemoveFav?nodeId='+ theNode);
      })
```

jQuery snippet

Oh, and I guess we need a little macro to list the favs too... well, for this, I decided to go back to my old school XSLT roots ;)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
  <!ENTITY nbsp "&#x00A0;">
  <!ENTITY times "&#215;">
]>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxml="urn:schemas-microsoft-com:xslt"
  xmlns:umbraco.library="urn:umbraco.library" xmlns:Exslt.ExsltCommon="urn:Exslt.ExsltCommon"
xmlns:Exslt.ExsltDatesAndTimes="urn:Exslt.ExsltDatesAndTimes" xmlns:Exslt.ExsltMath="urn:Exslt.ExsltMath"
xmlns:Exslt.ExsltRegularExpressions="urn:Exslt.ExsltRegularExpressions"
xmlns:Exslt.ExsltStrings="urn:Exslt.ExsltStrings" xmlns:Exslt.ExsltSets="urn:Exslt.ExsltSets"
xmlns:ucomponents.cms="urn:ucomponents.cms" xmlns:ucomponents.dates="urn:ucomponents.dates"
xmlns:ucomponents.email="urn:ucomponents.email" xmlns:ucomponents.io="urn:ucomponents.io"
xmlns:ucomponents.media="urn:ucomponents.media" xmlns:ucomponents.members="urn:ucomponents.members"
xmlns:ucomponents.nodes="urn:ucomponents.nodes" xmlns:ucomponents.random="urn:ucomponents.random"
xmlns:ucomponents.request="urn:ucomponents.request" xmlns:ucomponents.search="urn:ucomponents.search"
xmlns:ucomponents.strings="urn:ucomponents.strings" xmlns:ucomponents.urls="urn:ucomponents.urls"
xmlns:ucomponents.xml="urn:ucomponents.xml"
  exclude-result-prefixes="msxml umbraco.library Exslt.ExsltCommon Exslt.ExsltDatesAndTimes Exslt.ExsltMath
Exslt.ExsltRegularExpressions Exslt.ExsltStrings Exslt.ExsltSets ucomponents.cms ucomponents.dates ucomponents.email
ucomponents.io ucomponents.media ucomponents.members ucomponents.nodes ucomponents.random ucomponents.request
ucomponents.search ucomponents.strings ucomponents.urls ucomponents.xml ">

  <xsl:output method="xml" omit-xml-declaration="yes"/>

  <xsl:param name="currentPage"/>

  <xsl:template match="/">
    <xsl:variable name="theMember" select="umbraco.library:GetCurrentMember()" />
    <xsl:if test="$theMember">
      <div class="well well-small">
        <h4>My Favorites</h4>
        <div id="favList">
          <xsl:if test="$theMember/myFavs != ''">
            <ul class="unstyled">
              <xsl:for-each select="umbraco.library:Split($theMember/myFavs, ',')/value">
                <xsl:if test=". != ''">
                  <xsl:variable name="link" select="umbraco.library:NiceUrl(.)" />
                    <li>
                      <a href="#" rel="{.}" class="btn btn-mini btn-danger delete">&times;</a>&nbsp;&nbsp;
                      <a href="{$link}"><xsl:value-of select="umbraco.library:GetXmlNodeById(.)/@nodeName" /></a>
                    </li>
                </xsl:if>
              </xsl:for-each>
            </ul>
          </xsl:if>
        </div>
      </div>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

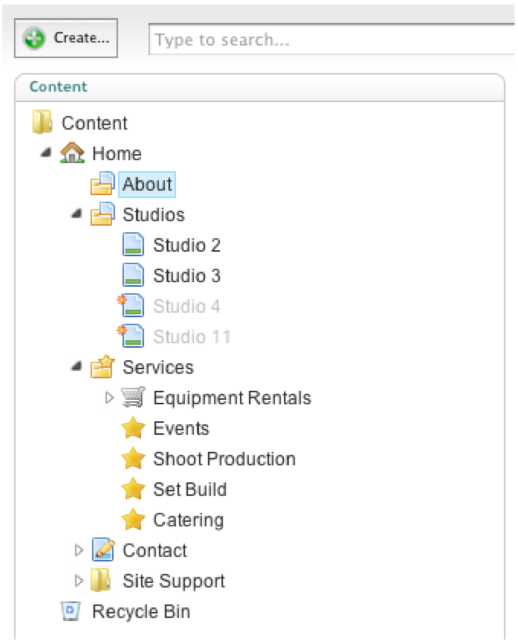
listMyFavs.xslt

With that we are just about done! There are many modifications and enhancements that I know all of you can make to this, so have fun and enjoy your holiday season!

AutoIcons, Conventions in Umbraco

— by Laurence Gillian

I'll make a slight assumption - you've used FamFamFam and played with Umbraco. Those icons make life easier for content editors, which makes us look better to our clients. Across the universe everyone wins.

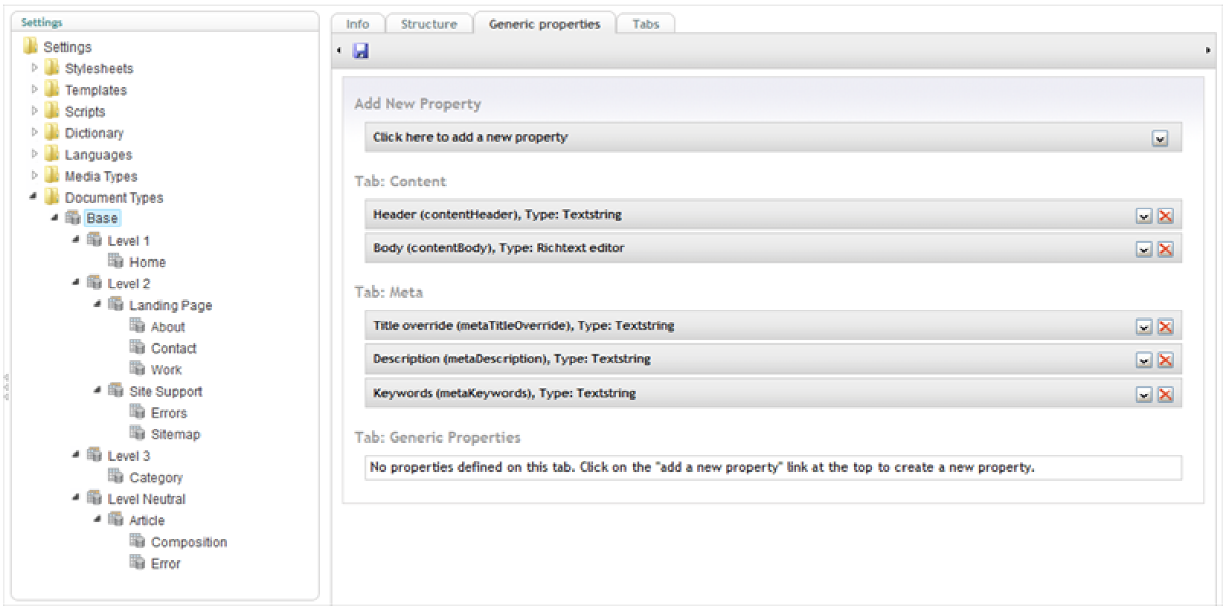


But, setting it up is a pain. There are 1003 icons, on average 15 used and scrolling that dropdown is - well horid. Even displaying a preview within Umbraco causes performance issues. To compound matters it gets worse, one day you're smoking your pipe (slippers) and login to a 'live' site. In a flash you realise you've used different icons between each site - it's confusing for users and frustrating for (anally retentive) developers like me.

So what's the solution?

Convention over Configuration

We use conventions when building our Umbraco websites, for example our DocTypes and PropertyAliases follow these patterns: (type prefix, etc)


Settings

- Settings
  - Stylesheets
  - Templates
  - Scripts
  - Dictionary
  - Languages
  - Media Types
  - Document Types
    - Base
      - Level 1
        - Home
      - Level 2
        - Landing Page
          - About
          - Contact
          - Work
        - Site Support
          - Errors
          - Sitemap
      - Level 3
        - Category
      - Level Neutral
        - Article
          - Composition
          - Error

Info | Structure | Generic properties | Tabs

Add New Property

Click here to add a new property

Tab: Content

Header (contentHeader), Type: Textstring

Body (contentBody), Type: Richtext editor

Tab: Meta

Title override (metaTitleOverride), Type: Textstring

Description (metaDescription), Type: Textstring

Keywords (metaKeywords), Type: Textstring

Tab: Generic Properties

No properties defined on this tab. Click on the "add a new property" link at the top to create a new property.

Everything inherits from Base and then we use some containers to logically group our document types, e.g. Level 1, Level 2, Level Neutral.

Something I've often dreamt is that Document Types lived as simple XML documents on the file system, editable via Umbraco or a editor of your choice. Document Types could support multiple parents (and children). These multiple points of entry could be rendered into the tree which would represent the data structure, completly removing the need for grouping Document Types.

This however is a dream. The reality I do know, from our agency and working with other people is most Umbraco developers use patterns. These feature in DocTypes, Properties, Macro Naming Conventions, etc. and are of colossal importance as getting architecture right (consistent) is one of the most important parts of a successful Umbraco build.

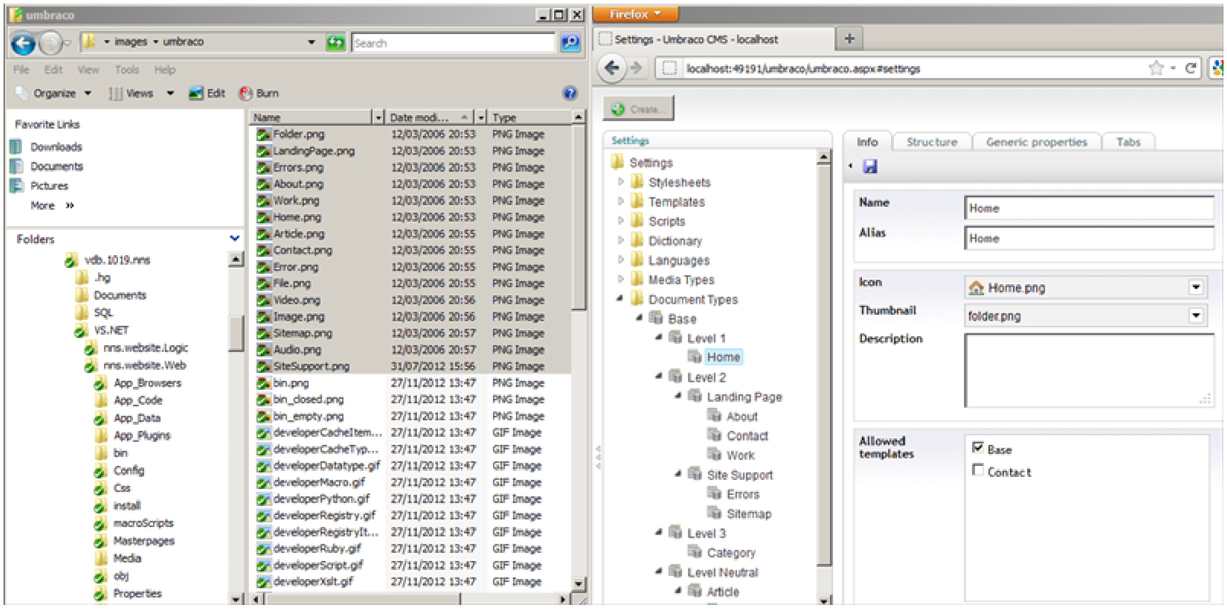
So a simple idea. Why not name our icons the same as our DocTypeAlias? The 'One to One' relationship would make finding them easier in that dropdown list and could save us some time.

Sounds like a convention to me

But wait a minute, that's still a manual task. So lets automate it! Earlier in the year myself and Hendy decided to write a package that does exactly that. It saves time and ensures sites are consistent.

So...

On application start-up, (if `UmbracoDebugMode = True`)
If an image matches DocTypeAlias (umbraco/images/umbraco)
Then set as icon.



Name	Date modi...	Type
Folder.png	12/03/2006 20:53	PNG Image
LandingPage.png	12/03/2006 20:53	PNG Image
Errors.png	12/03/2006 20:53	PNG Image
About.png	12/03/2006 20:53	PNG Image
Work.png	12/03/2006 20:53	PNG Image
Home.png	12/03/2006 20:53	PNG Image
Article.png	12/03/2006 20:55	PNG Image
Contact.png	12/03/2006 20:55	PNG Image
Error.png	12/03/2006 20:55	PNG Image
File.png	12/03/2006 20:55	PNG Image
Video.png	12/03/2006 20:56	PNG Image
Image.png	12/03/2006 20:56	PNG Image
Sitemap.png	12/03/2006 20:57	PNG Image
Audio.png	12/03/2006 20:57	PNG Image
SiteSupport.png	31/07/2012 15:56	PNG Image
bin.png	27/11/2012 13:47	PNG Image
bin_closed.png	27/11/2012 13:47	PNG Image
bin_empty.png	27/11/2012 13:47	PNG Image
developerCacheItem...	27/11/2012 13:47	GIF Image
developerCacheTyp...	27/11/2012 13:47	GIF Image
developerDatatype.gif	27/11/2012 13:47	GIF Image
developerMacro.gif	27/11/2012 13:47	GIF Image
developerPython.gif	27/11/2012 13:47	GIF Image
developerRegistry.gif	27/11/2012 13:47	GIF Image
developerRegistryIt...	27/11/2012 13:47	GIF Image
developerRuby.gif	27/11/2012 13:47	GIF Image
developerScript.gif	27/11/2012 13:47	GIF Image
developerXslt.gif	27/11/2012 13:47	GIF Image

Name: Home
Alias: Home
Icon: Home.png
Thumbnail: folder.png
Description:
Allowed templates: [✓ Base] Contact

Everytime the Umbraco application is restarted the code checks for new files and correctly wires them up! Like magic!

Summary

1. Package provides a default behaviour, that makes setting up Umbraco easier. :-)
2. Re-use a default collection of Icons across all websites.
3. Give your designer and architect access to Hg/Git, they can be part of the UI process. On larger sites which might have custom sections and icons this saves wasting developer time wiring up new icons.
4. Get the Package from Umbraco - Auto Icons
5. Psssst! This also support Media and Member Types too (:
6. @Hendy - wouldn't it be kewl, if we made it so it checked the umbracoExtension property/label and then overlayed a little symbol over the picked icon.

All the best for 2013,

Online Video Clip

Ideas

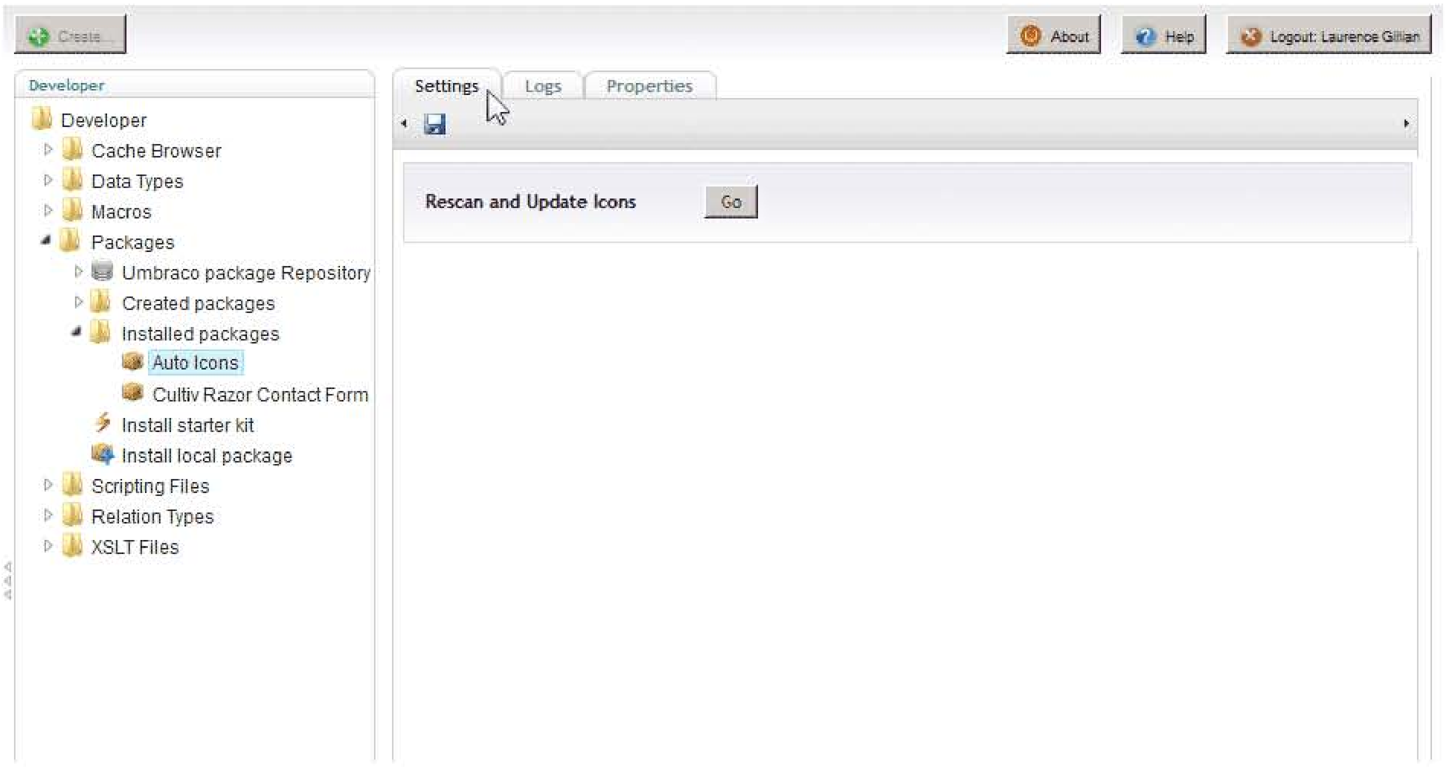
Package Configuration View

It would be neat if a package's configuration lived inside that package. There's no standard approach right now, with some packages adding a tabs whilst others add trees (uComponents is a good example). A consistent approach would be great and this should make life easier for package developers too.

Packages could become portable with no dependencies on dashboard.config. Neato!

This can be extended further with packages being 'picked' and implemented into existing trees. This would be great for packages like FALM and 301 URL Tracker that you may want certain User Groups to have access to.

On a bespoke project a developer could build custom functionality in a series of standalone packages which could then be loaded into a custom section & tree for certain users. It would be fairly straightforward to disable the 'properties' tab, except when being viewed through the Developer section.


Create
About
Help
Logout: Laurence Gillian
Developer
Developer
Cache Browser
Data Types
Macros
Packages
Umbraco package Repository
Created packages
Installed packages
Auto Icons
Cultiv Razor Contact Form
Install starter kit
Install local package
Scripting Files
Relation Types
XSLT Files
Settings
Logs
Properties
Rescan and Update Icons
Go

Quick Tips for Improving your Callouts

— by Tom Fulton

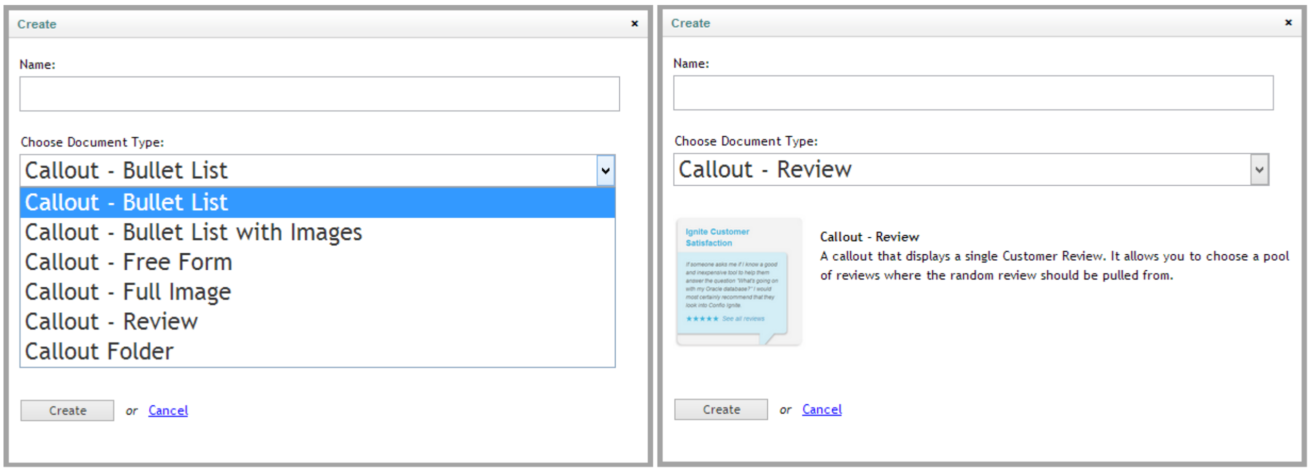
It seems like the majority of today's sites have some type of callouts (widgets) that need to be displayed on various pages, usually in a sidebar. Umbraco is great at handling this situation by utilizing document types and some of the new built-in datatypes. Hendy Racher wrote a great blog post a some time ago explaining the concept, but the gist is:

- Create a section outside of your content tree to manage Callouts via document types
- Use a Multi-Node Tree Picker on Content pages to select callouts to display
- Use a macro to loop through the selected callouts and render them accordingly

Here I'll detail a few quick tips we've implemented along the way to make the editing experience a bit more user-friendly and extensible. Some of these might be obvious to most, but hopefully you'll come away with at least one new idea!

Create document types for different types of callouts

Chances are your site has a few different "types" (or layouts) for callouts. Rather than try to cram many different fields into one document type, or shoehorning advanced layouts into a Richtext Editor, make it super-simple for editors by creating a document type for each layout. For some extra love, add a screenshot for the thumbnail of each type, so editors know exactly what they're creating.


Create ×

Name:

Choose Document Type:

Callout - Bullet List

Callout - Bullet List
Callout - Bullet List with Images
Callout - Free Form
Callout - Full Image
Callout - Review
Callout Folder

Create or Cancel

Create ×

Name:

Choose Document Type:

Callout - Review

Callout - Review
A callout that displays a single Customer Review. It allows you to choose a pool of reviews where the random review should be pulled from.

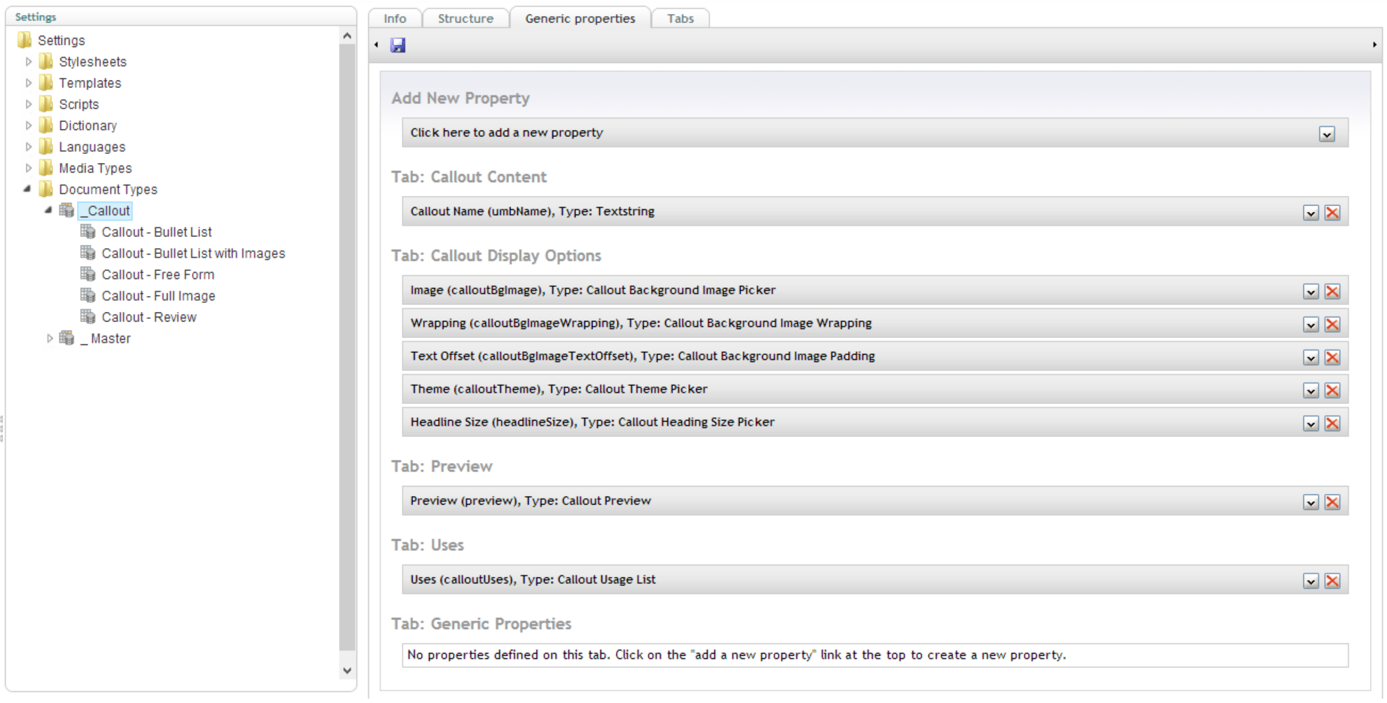
Create or Cancel

Using specific document types for differing layouts, and screenshots for thumbnails

Each specific document type should have fields relevant only to their layout. For example, the "Bullet List" callout might use a Multiple Textstring datatype, while the "Review" callout would use a picker to select from available reviews.

Use a base document type for global fields

Even with different types of callouts, you'll likely have a set of "global" options that apply across all types. A great way to handle this is to create a Master document type for callouts, that all your Callout document types inherit from.



Add New Property

Click here to add a new property

Tab: Callout Content

Callout Name (umbName), Type: Textstring

Tab: Callout Display Options

Image (calloutBgImage), Type: Callout Background Image Picker

Wrapping (calloutBgImageWrapping), Type: Callout Background Image Wrapping

Text Offset (calloutBgImageTextOffset), Type: Callout Background Image Padding

Theme (calloutTheme), Type: Callout Theme Picker

Headline Size (headlineSize), Type: Callout Heading Size Picker

Tab: Preview

Preview (preview), Type: Callout Preview

Tab: Uses

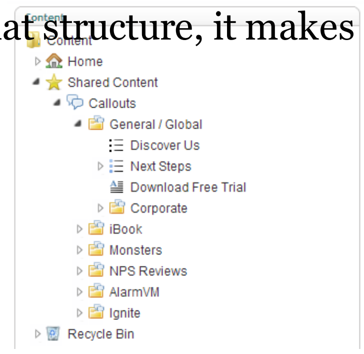
Uses (calloutUses), Type: Callout Usage List

Tab: Generic Properties

No properties defined on this tab. Click on the "add a new property" link at the top to create a new property.

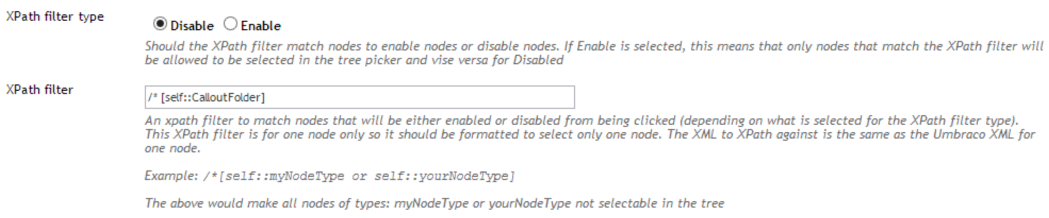
Using a Master document type for global callout properties

Organize callouts into folders



In large sites, your "Callouts" area can get extremely long. In a flat structure, it makes it pretty difficult for editors to find anything. To alleviate this, create a new document type called "Callout Folder", allow Callouts to be created beneath it, and allow it to be created under the "Callouts" node. Then, you can end up with a nice structure as shown in the figure.

To ensure a Callout Folder can't be selected from a content page, adjust your MNTP settings to disallow this:



XPath filter type [✓ Disable] Enable

Should the XPath filter match nodes to enable nodes or disable nodes. If Enable is selected, this means that only nodes that match the XPath filter will be allowed to be selected in the tree picker and vise versa for Disabled

XPath filter `/*[self::CalloutFolder]`

An xpath filter to match nodes that will be either enabled or disabled from being clicked (depending on what is selected for the XPath filter type). This XPath filter is for one node only so it should be formatted to select only one node. The XML to XPath against is the same as the Umbraco XML for one node.

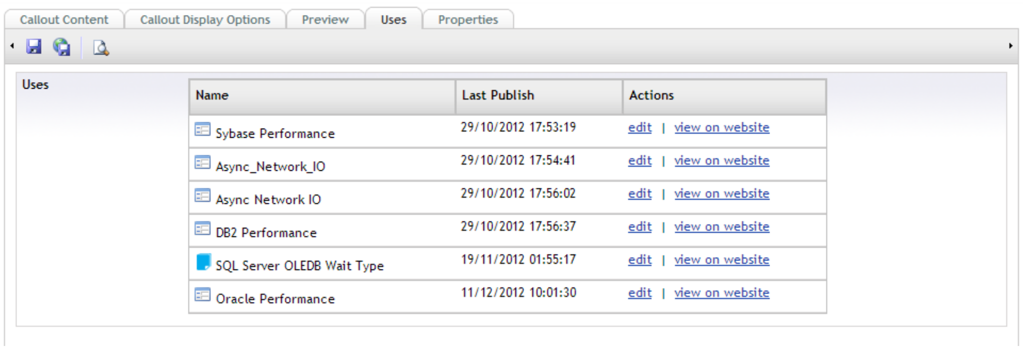
Example: /[self::myNodeType or self::yourNodeType]*

The above would make all nodes of types: myNodeType or yourNodeType not selectable in the tree

Keeping track of callout usages

There's a nice hidden gem in Umbraco 4.9 called Picker Relations, which wires up a Multi-Node Tree Picker (or other picker datatypes) to Relations. This means that when a node (Callout) is selected, a Relation entry is automatically created when the document is saved.

This gives us some really cool possibilities - in this case, we can use it to show all nodes where a given callout is referenced, making it super easy for editors to know exactly which pages they're affecting when making an edit. *No code required!*



Uses

Name	Last Publish	Actions
Sybase Performance	29/10/2012 17:53:19	edit | view on website
Async_Network_IO	29/10/2012 17:54:41	edit | view on website
Async Network IO	29/10/2012 17:56:02	edit | view on website
DB2 Performance	29/10/2012 17:56:37	edit | view on website
SQL Server OLEDB Wait Type	19/11/2012 01:55:17	edit | view on website
Oracle Performance	11/12/2012 10:01:30	edit | view on website

Showing which pages a callout is selected on

To list the related pages, there's another built-in datatype called Relation Links, which does all the work for you (a custom datatype is pictured above). For more details on configuring these datatypes, check out Hendy Racher's post: Who Picked

This?

So, I hope you've got at least one new idea to improve your Callout system for yourself and content editors. Merry Christmas!

Wrapping Up: Building this Site

— by Chriztian Steinmeier

Hi, my name is Chriztian; I'm a Frontend Developer at Vokseværk where I work almost exclusively with Umbraco websites every day. It's total bliss :-)

How this came to be

I registered this domain a couple of years ago, thinking it would be cool to do some kind of calendar thing with it.

This year I finally got the right idea (in time, too) and started putting it together. This post will highlight a couple of things from the experience that I think would be of interest to you...

#MadeWithPaper

Renaming the /umbraco folder

Hey, don't do it unless you *really* have to - I'm pedantic about some things, which meant that because I wanted the calendar to be on the `24days.in/umbraco/` URL (which is usually where you log in to the Back-office), I had to do it. Let me repeat: *Don't do it* - Umbraco is not built for that yet. I had to literally do "search and replace" through the Umbraco source code to find some of the places where that URL was hard-coded.

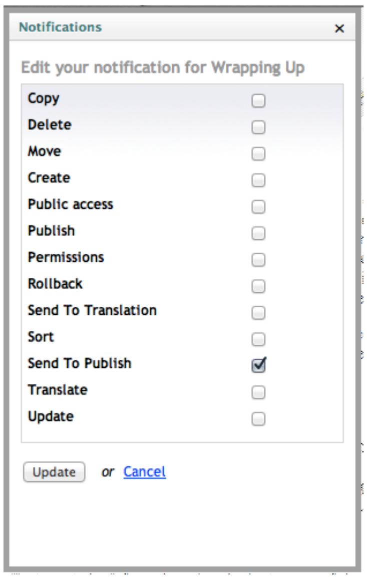
Yes, there *is* a key in `web.config` named `umbracoPath` which *does* tell some parts of the system where Umbraco is installed, but it's *not* used everywhere so you're going to do some manual tweaking to get it to work. And *yes*, some of it can be "fixed" using URL rewrites, but...

Even though you've sent that folder back in time - to way before it was created - as soon as you install a package that is supposed to put files somewhere below it (could be icons for **DocTypes** or some code in the `/umbraco/plugins` folder) you'll have the umbraco folder back, safe and sound in 2012!

Worst part though: uComponents' keyboard navigation in the trees didn't work anymore, which was a major pain - I heavily rely on the arrow keys when I'm walking around in all of the trees of the Back-office! Thank God, Lee Kelleher got me a patched DLL with a fix for that bug - *#h5yr*, Lee.

Notifications

Umbraco has a built-in system for notifications with regard to a content node, so you can easily be notified whenever an editor performs a specific action on the node. On this project every author is a User in the system, with access to only their own article node, and a designated media folder. They are not allowed to Publish, but only to "Send for approval" as it's called. This way I just got an email whenever an author deemed their article ready for proofreading and publishing.



Notifications ×

Edit your notification for Wrapping Up

Copy	☐
Delete	☐
Move	☐
Create	☐
Public access	☐
Publish	☐
Permissions	☐
Rollback	☐
Send To Translation	☐
Sort	☐
Send To Publish	✓
Translate	☐
Update	☐

Update or Cancel

Notifications, built-in

Note: To get notifications to work, you need to configure the `<smtp>` section in `web.config`, and set an email address under the `<notifications>` setting in `umbracoSettings.config`.

Code Snippets

I knew from the start that many articles would contain code snippets and I really wanted to make them easy to manage. We used the awesome Prism for doing the syntax highlighting but still, I knew working with them in the WYSIWYG was going to be a pain. I did **not** want to ask authors to convert their code samples to any kind of HTML which would ensure we'd have loads of possibilities for errors in them.

Fortunately I found a simple but powerful solution using a macro and (of course) an XSLT extension from uComponents:

You create a Code Snippet in the Media section, upload your file to it and insert it in the WYSIWYG with the macro - here's the relevant XSLT:

```
<xsl:template match="CodeSnippet">
    <xsl:variable name="lang" select="$lexer[@ext = current()/umbracoExtension]" />
    <pre class="language-{$lang}"><code>
        <xsl:value-of select="ucom.IO:LoadFile(umbracoFile)" />
    </code></pre>
    <xsl:apply-templates select="caption[normalize-space()]" />
</xsl:template>
```

The `LoadFile()` method simply does that (loads a file) and when we use an XSLT `value-of` instruction, the processor will do all the output-escaping necessary for especially HTML, XML & XSLT samples to "just work" with no preprocessing.

(The `$Lexer` variable serves as a proxy for choosing the file's language based on its file extension, because some files needed to be mapped to a specific language.)

Using a macro also had the fantastic sideeffect of an author being able to update his codesnippets just by uploading a new file. If they were just content in the WYSIWYG they'd have to wait for me to actually publish their changes.

Thank you, thank you, thank you

I'd really like to thank a few people, so here goes:

Initially, I figured I'd alert HQ about this, just to be sure I wasn't botching one of their secret plans, or misusing the Umbraco name for something they didn't comply with. Fortunately they thought it was an excellent idea, and encouraged me to go ahead, so ***thanks for that Niels!***

I also want to thank Jan and Sebastian for instantly agreeing to do this when asked if they'd like to team up and do an "Umbraco themed" christmas calendar. This wouldn't have happened had they not agreed, thereby encouraging me to go ahead. ***Thanks guys!***

Finally, a huge ***thank you*** to all the authors for joining the project - it's been a joy reading another surprising article day after day, *#h5yr* to all of you!

Now go have a Merry Christmas

We're going to release the source code of this site on GitHub, but **not** until we've all had a nice break from coding and what other things we do everyday, so have a lovely Christmas and thank you **very** much for following along this month.

/Chriztian