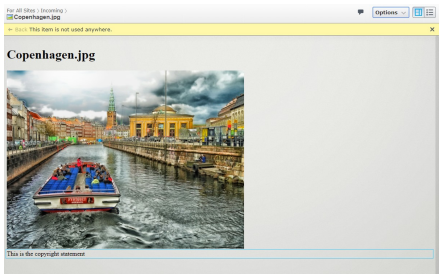
ALLAN THRAEN | 5 years ago | PDF |

Tips and Tricks CMS Optimizely (Episerver)

ON PAGE EDIT FOR IMAGES



The default image edit mode in Episerver CMS is a little bit boring - it's nothing but an image tag with the actual image. Why not offer a richer on page edit experience for image media as you typically get for pages and blocks?

In Episerver you get pretty used to building TemplateDescriptor - typically Controller classes that deal with how a specific content type should be controlled and rendered. It can also be other stuff - like full HttpHandlers with a [TemplateDescriptor] attribute, but that's another story :-)

Anyway - since you typically have to write all of the template descriptors yourself, you might not always realize just how easy it is to replace the few built-in default controllers available. For example, Episerver comes out-of-the-box with Media Edit controllers (for images, video and other media). They are located in the namespace EPiServer.Cms.Shell.UI.Controllers.Preview.Internal and are extremely basic - for images for instance it does nothing but render out the "<img>" tag.

Decompiling it, it looks like this:

```
[TemplateDescriptor(AvailableWithoutTag = false, Inherited = true, TagString = "Edit", TemplateTypeCa
  public class ImageEditController : MediaEditController<IContentImage>
  {
    private readonly UrlResolver _urlResolver;
    private readonly TemplateResolver _templateResolver;

    public ImageEditController()
          : this(ServiceLocator.Current.GetInstance<UrlResolver>(), ServiceLocator.Current.GetInstanc

    }

    public ImageEditController(UrlResolver urlResolver, TemplateResolver templateResolver)
                                                                                          {
      this._urlResolver = urlResolver;
      this._templateResolver = templateResolver;
    }

    protected override string GetPreviewContent(IContentImage content)
                                                                 {
      return new TagBuilder("img")
      {
        Attributes = {
          {
            "src",
            content.PreviewUrl(this._urlResolver, this._templateResolver)
          },
          {
            "alt",
            content.Name
          }
        }
      }.ToString(TagRenderMode.SelfClosing);
    }
  }
```

Note: When dealing with media there are 3 different kinds of template descriptors out the box:

1. The HttpHandler that works across all rendering tags and will return the actual binary file (based on EPiServer.Web.Internal.ContentMediaHttpHandler)
2. The HttpHandler for Preview (which is somewhat similar to #1, but with different caching).
3. The MediaEditController (and classes inheriting from it) that will return the HTML for the On Page Edit.

For now, we want to replace #3 with another that actually gives us real on-page-edit capabilities and not just a static image.

Now, since there already exist controllers, to deal with this, we need to override them. Luckily, there is great documentation available that describe in which order template descriptors are picked. Based on this order, we could either hook into an event and pick the handler we want, or we could just make sure it's earlier in the hierachy. For instance by putting it closer in the inheritance chain; The default one maps to IContentImage - we'll instead use ImageData (which implements IContentImage, and which for instance the ImageFile in Alloy is based on).

We'll of course make sure that it's only uses this code for the Edit rendering tag and then we can make a completely standard template controller and view.

```
1  using System.Linq;
2  using System.Web.Mvc;
3  using EPiServer.Core;
4  using EPiServer.Framework.DataAnnotations;
5  using EPiServer.Framework.Web;
6  using SmallExperiments.Business;
7  using SmallExperiments.Models.Pages;
8  using SmallExperiments.Models.ViewModels;
9  using EPiServer.Web;
10 using EPiServer.Web.Mvc;
11 using EPiServer;
12 using EPiServer.Framework.Web.Mvc;
```

```cs
using SmallExperiments.Models.Media;
using EPiServer.Shell;
using EPiServer.Web.Routing;
using EPiServer.ServiceLocation;
using EPiServer.Cms.Shell;

namespace SmallExperiments.Controllers
{
    [TemplateDescriptor(
        Inherited = true,
        TemplateTypeCategory = TemplateTypeCategories.MvcController,
        Tags = new[] { RenderingTags.Edit },
        AvailableWithoutTag = false,ModelType = typeof(ImageData))]
    [VisitorGroupImpersonation]
    [RequireClientResources]
    public class ImagePreviewController : ActionControllerBase, IRenderTemplate<ImageData>, IRenderTemplate
    {
        private readonly UrlResolver _urlResolver;
        private readonly TemplateResolver _templateResolver;

        public ImagePreviewController()
    : this(ServiceLocator.Current.GetInstance<UrlResolver>(), ServiceLocator.Current.GetInstance<TemplateResolver>())
        {
        }

        public ImagePreviewController(UrlResolver urlResolver, TemplateResolver templateResolver)
        {
            this._urlResolver = urlResolver;
            this._templateResolver = templateResolver;
        }

        public ActionResult Index(IContent currentContent)
        {
            var url = (currentContent as IContentImage).PreviewUrl(_urlResolver, _templateResolver);
            ViewBag.ImageUrl = url;
            return View(currentContent);
        }
    }


}
```

ImagePreviewController.cs hosted with ♥ by GitHub view raw

```cshtml
@using SmallExperiments
@model SmallExperiments.Models.Media.ImageFile
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title></title>
</head>
<body>
    <div>
        <h1>@Html.PropertyFor(m => m.Name)</h1>
        <img src="@ViewBag.ImageUrl" width="600" />

        @Html.PropertyFor(m => m.Copyright)
    </div>
</body>
</html>
```

Index.cshtml hosted with ♥ by GitHub view raw

Tips and Tricks | CMS | Optimizely (Episerver)

RECENT POSTS

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688