← Blog posts

Understanding Episerver CMS - How are images rendered

CMS, TIPS AND TRICKS, OPTIMIZELY (EPISERVER)

Allan Thraen | 3 Years Ago | PDF |

When using reusable content such as images, the actual HTML rendering of them can happen multiple places. But when is what used? And how can you customize it?

First, a little disclaimer: There are lots and lots of both blog posts and forum posts on various aspects of this topic. This is merely yet-another, but I wrote it to have a summarized overview of the different ways images get rendered in Episerver CMS.

Generally speaking, there usually 5 ways images gets rendered:

1. **An image property rendered using Html.PropertyFor()** in Razor. If the property is a ContentReference with a [UIHint(UIHint.Image)] it will try to use the Image Display template if one exist (typically found in ~/Views/Shared/DisplayTemplates/Image.cshtml).
2. **An image property rendered in a custom way.** Often the view will have code like but it could vary.
3. **Image rendered as a IContent** (block style) in a Content Area. This is usually done with a Controller (inheriting from PartialContentController<ImageFile> as in Alloy) and a corresponding View. Maybe even a view model.
4. **Image rendered as Partial Content** in a Content Area. I believe this is a fairly rare approach (and it's not enabled in Alloy by default), but it can be quite handy to setup multiple renderings of images for different areas named by a tag. It doesn't use a controller, so it should be a bit lighter on performance. It requires you to set up an IViewTemplateModelRegistrator class and register it like this:

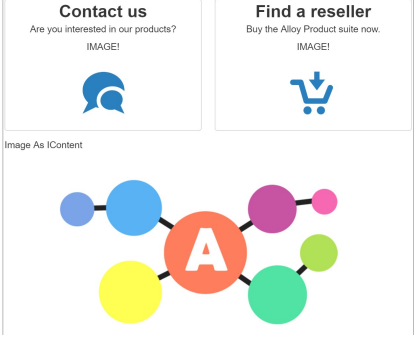
```
viewTemplateModelRegistrator.Add(typeof(ImageFile), new TemplateModel
{
    Name="ImageNarrow",
    Tags = new[] {Global.ContentAreaTags.OneThirdWidth},
    AvailableWithoutTag=false,
    Path="~/Views/ImageFile/Index2.cshtml"
});
```

   Note, that I here have registrered it only for one specific tag. If a controller exist that covers the same, it will override.
5. **Image rendered in an XHtmlField (by TinyMCE).** The default behavior when you drag an image into the TinyMCE editor (or add it using the buttons) is that TinyMCE treats it as an image (surprise!). Since it's possible in the editor to do simple alterations to the image like setting width and height for the image tag, the editor controls how the image is rendered. If you want your image to always be rendered in a special way (perhaps wrapped in a styled div, with an image text, as a thumbnail or with a specific cool overlay) the easiest approach is to change the DropBehavior when an image is dropped on the XHTML Field, to make it work as if you are dropping a block (which in turn you can control through #3 or #4). You can achieve that this way (also described here):

```
[UIDescriptorRegistration]
public class ImageUIDescriptor : UIDescriptor<ImageFile>, IEditorDropBehavior
{
    public EditorDropBehavior EditorDropBehaviour { get; set; }

    public ImageUIDescriptor()
                        {
        EditorDropBehaviour = EditorDropBehavior.CreateContentBlock;
    }
}
```

   Alternatively you could plugin and runtime iterate the fragments your xhtml field consist of and replace the image with your own code, or make a hidden plugin to TinyMCE that listens to the events when an image is added. But both of those seems a bit far fetched.



Consider the ContentArea above. Here, you see multiple approaches on how an actual image is rendered. The top block is a Teaser block, and it has a custom rendering or the image (#2).

The two narrow teasers below are also Image properties, but using rendered using Html.PropertyFor as in #1. They use the Image display template found in the Views/Shared/DisplayTemplates folder in Alloy.

The bottom image is simply dropped as a block on the content area directly and hence is displayed through a controller and a view as in #4. You'll notice that I've added a little text before the <IMG> tag for both the displaytemplates and the regular Controller/View to tell them apart. (I'll probably have to remember to remove that before release :-)).

Some additional fun facts about images

You might be aware, that Images inheriting from ImageData always have a Thumbnail blob property as well as the main blob for the image. And that you can access it by appending /Thumbnail to the image url.
And from there it's not hard to figure out that you can add as many of these alternative Blobs as you want on the image and call them accordingly (but be aware that if you are looking for automatic image resizing, you should generally use something like ImageResizer with disk caching(!) or a good DAM that's integrated). **But** did you know that you can also call /Download on an image url and it will be downloaded instead of shown?

CMS TIPS AND TRICKS OPTIMIZELY (EPISERVER)

Post Comments 0