ALLAN THRAEN | 6 years ago | PDF |

Digizuite Optimizely (Episerver) Addon Development Integrations

DIGIZUITE: KEEPING DEVELOPERS IN MIND WHEN BUILDING THE ADDON

I have worked on many different addon's for Episerver over the years - and used many more. One thing that often strikes me is that either an Addon is for editors or it is for developers, but rarely both. With the new Digizuite integration we are trying to give both groups the tools they need.

As I wrote in an earlier blog post, I'm currently helping Digizuite (DAM provider) with a connector for Episerver.

The main purpose of such a connector is (as with many other addons to Episerver) of course editorial. Editors and marketers should get instant and easy access to all the binary assets (images, videos, whitepapers and the like) that their company stores in the Digital Asset Management system - and as such there is of course a wide editor / UX focus in such an integration.

But at the same time, it's essential that we keep in mind exactly how Episerver is used as a platform for building great web sites. Typically, you have developers and web designers building a site exactly as they want it - supported by the CMS. And I believe the trick is to always keep them in mind when building new addons - because no matter how great an add-on you make, it will always end up being extended and customized to fit the specific needs of a specific web site. The key is to provide developers with the right set of tools to take advantage of the functionalities offered by the addon - and not just focus on the editors.

Content Modelling

An essential thing on any web site is the content modelling - which content types should there be, and which metadata (properties) should they contain. In Episerver having your content models as strongly typed code has been the de facto standard ever since Joel started doing it with Page Type Builder, back in 2010'ish - and has been part of the core product since version 7 (if I recall correctly).

So, while the addon of course should work out-of-the-box and hence has default asset content type definitions included, it should be both possible and easy for a developer to customize this and add their own content types, mapped to the corresponding asset types in Digizuite.

To make this process easy, we've simply introduced a couple of useful attributes for mapping content types as seen here:

```
[ContentType(GUID = "0A89E464-56D4-449F-AEA8-2BF774AB8730")]
[MediaDescriptor(ExtensionString = "jpg,jpeg,jpe,ico,gif,bmp,png")]
[DigizuiteType(AssetTypeIds = new[] { 4 })]
public class ImageFile : ImageData, IDigizuiteImage
{

    //Digizuite properties
    [UIHint(UIHint.Textarea)]
    public virtual string Description { get; set; }

    [Editable(false)]
    public virtual int ItemId { get; set; }

    [Editable(false)]
    public virtual int AssetType { get; set; }

    [Editable(false)]
    public virtual string CropName { get; set; }

    [UIHint(UIHint.Textarea)]
    public virtual string Note { get; set; }

    [Editable(false)]
    public virtual IList<ContentReference> Folders { get; set; }

    [UIHint(UIHint.Image)]
    [Editable(false)]
    public virtual ContentReference ParentImage { get; set; }

    [Editable(false)]
    public virtual int ImageWidth { get; set; }
    [Editable(false)]
    public virtual int ImageHeight { get; set; }


    [DigizuiteProperty(Name = "Software", MetaFieldLabelId = 11276)]
    public virtual string ExifSoftware { get; set; }
```

```
        [DigizuiteMediaFormat(GlobalConstants.MediaFormatNames.Image_JPGmedium)]
        [ScaffoldColumn(false)]
        public virtual Blob Medium { get; set; }

        [DigizuiteMediaFormat(GlobalConstants.MediaFormatNames.Image_JPGsmall)]
        [ScaffoldColumn(false)]
        public virtual Blob Small { get; set; }

        [ScaffoldColumn(false)]
        [DigizuiteProperty(Name="downloadSource")]
        [Editable(false)]
        public virtual string DownloadSource { get; set; }

        [Editable(false)]
        public virtual int AssetSize { get; set; }

    }
}
```

"DigizuiteType" maps this class to Digizuite Images (Digizuite ID 4). We also need to make sure we implement the interface IDigizuiteImage which adds a number of required properties needed for the connection (like the Digizuite ItemId, Asset Type ID and a few others).

Since it's both possible and easy to extend with custom meta-data on any asset type in Digizuite (and there is also a long list of already existing meta-data) developers can pick which they want to expose in Episerver and simply add the properties with a "DigizuiteProperty" attribute mapping it to Digizuite. This mapping ensures both read and write capabilities where possible.

Finally, in Digizuite you can define a number of Mediaformats (for multiple different kinds of media - for images, it's by default various sizes of JPG's and a transparent PNG). These are automatically generated, so you can ensure standardized sizes and resolutions used throughout the site.

By adding "DigizuiteMediaFormat" attribute to a Blob property you can map a given format directly to a blob, making it accessible by calling the Image Url +"/BlobName". Of course, if you don't do that, the mediaformats can still be accessed through a query parameter to the blobs.

Validation attributes

To make transitioning to a DAM much smoother, Digizuite Images essentially work exactly like Episerver images - and can for instance be inserted into ContentReference properties on content with a UIHint(UIHint.Image) attribute.

But sometimes editors don't know exactly what kind of image the designers had in mind for a given property. To help ensuring the quality of the site, we're adding validation attributes that can be applied to content models - like "RequireCrop" that makes sure that only images cropped in a certain way, or "RequireHasCrop" (only images that has a certain crop sibling) can be used in a given field. That way, if you have a placement for a banner, you can make sure that only a banner can fit in there.

```
    public class ProductPage : StandardPage, IHasRelatedContent
    {
        [ImageSize(MinWidth =800, Orientation =ImageOrientation.Portrait)]
        [UIHint(UIHint.Image)]
        public virtual ContentReference MainImage { get; set; }

        [UIHint(UIHint.Image)]
        [RequireCrop("Banner")]
        public virtual ContentReference BannerImage { get; set; }

        [UIHint(UIHint.Image)]
        [RequireHasCrop("WideScreen")]
        public virtual ContentReference ImageWithWidescreen { get; set; }

        [UIHint(UIHint.Image)]
        [DigizuiteMediaFormat(GlobalConstants.MediaFormatIds.Image_JPGbig)]
        public virtual ContentReference ImageFormatBig { get; set; }

//...
}
```

Since we now also now know the size of the image, we can also validate that only images with a certain minimum size is used on a property (to avoid those terribly upscaled images that should have been used as icons), simply by setting ImageSize attribute. Same attribute also let's you set a required image orientation (square, portrait, landscape).

We are also experimenting with adding a MediaFormat attribute to a content reference property will ensure that the image is always displayed using that media format.

Extension methods and IoC

We all know that developers will eventually be customizing the solutions much more than we can anticipate. And again - the best thing we can do is to make it as easy as possible. So we are also giving a lot of thought into adding extension methods that will make it easy to get as much as possible out of the new functionality.

A good example of that is that you can call "IDigizuiteImage.GetCrops()" on any Digizuite Image and get a list of assets that are crops of the original image.

Similarly, we are registering everything as services, making it very easy to inherit, customize, extend and replace all major significant parts of the integration if needed.

MediaUrlSegmentProviders

A lot of SEO experts seems to put a lot of emphasis on the URL segments in pretty much any url. Also, the URL's to documents and images. By default Episerver simply uses the name of the media asset, but we decided to go the extra mile and make a MediaUrlSegmentProvider model, so developers can customize exactly how they want the URL's to be generated and look like for the assets. Of course, we provide a default implementation - but if you want more, it's as simple as registering a new using the ServiceConfiguration attribute.

Tag Mapping

Finally, we are also looking into ways of letting developers and designers map Episervers rendering tags

to mediaformats and crops in Digizuite.

For instance - if an asset is rendered in a sidebar content area, with a 'sidebar' tag it would be nice to ensure that the crop used will look appropriate. Or - if a mobile rendering tag is present that a smaller version is transmitted, rather than the original 200 Mb Tiff :-)

We're still working one exactly how that code should work, but I'll share more details soon.

These are just some of the developer friendly approaches we have taken so far. I'd love to hear your thoughts, so let me know what you think!

Digizuite | Optimizely (Episerver) | Addon Development | Integrations

Introducing Digizuite DAM for Episerver

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