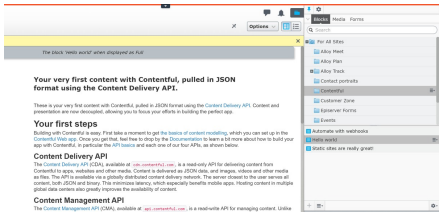
ALLAN THRAEN | 5 years ago | PDF |

Vision Demos & Prototypes Integrations Optimizely (Episerver) Contentful CMS

DISTRIBUTED CONTENT: DELIVERING AN EPISERVER WEB EXPERIENCE WITH CONTENTFUL CONTENT



The move in the market towards headless could also be seen as a tendency towards a deeper decoupling between content and experience delivery. Inspired by a few discussions, I've tried my hands on an uncommon combination: Contentful providing content delivered through an Episerver web experience layer.

For quite a while there has been a lot of focus on the so-called 'move' in the CMS market towards headless CMS, a CMS with an editorial backend and some kind of developer API (typically REST) instead of a front end. A move that Episerver followed with the Content Delivery API. The advantage of course being that developers can build whatever frontend they like in their preferred language, for any channel - going from web and phones to apps, smart-watches, billboards, print and so on.

Typically, though a headless CMS is focused around 'pure' content management though - and when you consider a system like Episerver today, all the main sellable features are around experience management on the web delivery channel (I include mobile web here as well).

With this in mind I tend to agree with some of the thoughts shared by fellow EMVP Deane Barker here: The Future Might be Distributed. A topic he also did a great presentation on in early february '19 at the Episerver Partner Close-up event.

I can definitely see cases where it will make great sense to pull content from various providers and distribute them out through a number of delivery/experience channels managed in different experience management platforms. In fact, I would argue that it already happens massively today (and has done for a while) if we simply broaden the "content" concept a bit:

- DAM systems (like my friends @ Digizuite) manage and deliver binary content (with metadata) to many different consumers (Episerver being one) that then create and manage channel experiences with it.
- PIM systems deliver structured product content to many consumers, Episerver Commerce included
- Syndication hubs as our very own Episerver world aggregates and syndicates content produced by many other different content hubs, through rss/atom syndication (which is probably how you found this blog)
- And historically I certainly still remember all the hours I've wasted on CMIS which was supposed to be the standard that would change the world when it came sharing content between platforms (but that's another story all together best explained here).

Episerver Content Providers

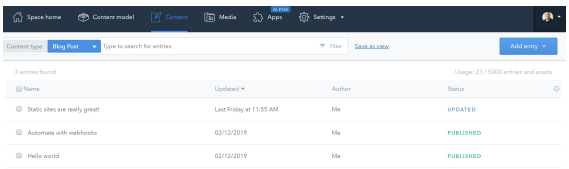
For more than 10 years Episerver CMS has had a neat little feature known as Content Providers. And if you browse around my blog you'll see I have historically written many examples of them - and this seemed like a perfect opportunity to do that again.

I recently spend some time getting to know Contentful a SaaS based Contenthub that's pretty neat for pure content management with a powerful API. I figured this would be a great opportunity to build a simple integration that would mimick how you could use Episerver to orchestrate Contentful content on a web channel(see screenshot above).

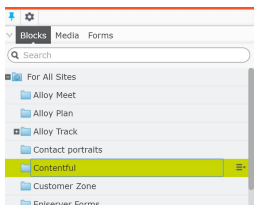
I signed up for a free account, added the proper nuget package to my Episerver solution and started to code a pretty basic Content Provider. You can find the code in a gist below.

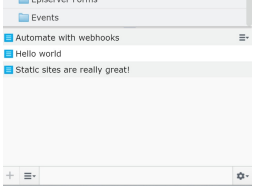
In the initialization I map up a custom block model I created in Episerver to the Contentful content model for a blog post, and in my web.config I have configured the content provider and attached it to an entrypoint in my Block folders.

Here is a filtered list of my Contentful content (blog posts):



And here I have my node with the blog posts automagically appearing below.





Disclaimer: This code is only a PoC to show how the distributed scenario discussed could work in real life. Currently the provider is read-only, single-language, not security aware and with no built-in resilience. Feel free to be inspired, but use at your own risk.

Also, feel free to drop me a line in the comments below and let me know how you think the systems will play together in a distributed future!

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.DataAnnotations;
using EPiServer.Web;

namespace ContentfulDemo.Models.Blocks
{
    [ContentType(DisplayName = "Contentful Blog Block", GUID = "f28b3914-509d-41e4-bc8c-da0f63065fd7", Description = "")]
    public class CFBlogBlock : BlockData
    {
        [Editable(false)]
        public virtual string Name { get; set; }

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

        [Editable(false)]
        [UIHint(UIHint.Textarea)]
        public virtual string Description { get; set; }

        [Editable(false)]
        [UIHint(UIHint.Textarea)]
        public virtual string Body { get; set; }


    }
}
```

CFBlogBlock.cs hosted with ♥ by GitHub view raw

```
@using EPiServer.Core
@using EPiServer.Web.Mvc.Html

@model ContentfulDemo.Models.Blocks.CFBlogBlock

<div class="row">
    <div class="span4 hidden-tablet hidden-phone">
        <img src="@Model.HeroImage"/>
    </div>

    <div class="span8">
        <h1 class="jumbotron">@Model.Name</h1>
        <p class="subHeader">@Model.Description</p>
        <p>@Html.Raw(Markdig.Markdown.ToHtml(Model.Body))</p>
    </div>
</div>
```

CFBlogBlock.cshtml hosted with ♥ by GitHub view raw

```
using Contentful.Core;
using ContentfulDemo.Contentful;
using ContentfulDemo.Models.Blocks;
using EPiServer;
using EPiServer.Construction;
using EPiServer.Core;
using EPiServer.Data.Entity;
using EPiServer.DataAbstraction;
using EPiServer.ServiceLocation;
using EPiServer.Web;
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Globalization;
using System.Linq;
using System.Net.Http;
using System.Web;
using System.Web.Configuration;

namespace ContentfulDemo.ContentfulTools
{
    public class ContentfulContentProvider : ContentProvider
    {
        protected IdentityMappingService IdentityMappingService { get; set; }
        protected ExternalTypeMapper TypeMapper { get; set; }
        protected IContentTypeRepository TypeRepo { get; set; }
        protected IContentFactory _ContentFactory { get; set; }
        protected IContentRepository ContentRepo { get; set; }

        public const string KEY = "CF";
        private const string BASEURI = "CF://"+KEY+"/";

        private ContentfulClient _Client { get; set; }

        public ContentfulContentProvider(IdentityMappingService mappingService,ExternalTypeMapper mapper, IContentTypeRepository typerepo, IContentFactor
        {
            IdentityMappingService = mappingService;
            TypeMapper = mapper;
            TypeRepo = typerepo;
            _ContentFactory = contentfactory;
            ContentRepo = contentrepo;
        }

        protected override IList<GetChildrenReferenceResult> LoadChildrenReferencesAndTypes(ContentReference contentLink, string languageID, out bool languag
        {
            languageSpecific = false;
            if (contentLink.CompareToIgnoreWorkID(EntryPoint))
            {
```

```
            //Root, list mapped types
            //Get list of all mapped items
            var results=_Client.GetEntries<dynamic>().Result;
            List<GetChildrenReferenceResult> rt = new List<GetChildrenReferenceResult>();
            foreach(var itm in results.Items)
            {
               string ctid = itm.sys.contentType.sys.id;
               if (TypeMapper.TypeMappings.ContainsKey(ctid))
               {
                  string entryid = itm.sys.id;
                  var id=IdentityMappingService.Get(new Uri(new Uri(BASEURI), entryid), true);

                  rt.Add(new GetChildrenReferenceResult() { ContentLink = id.ContentLink, IsLeafNode = true, ModelType = TypeMapper.TypeMappings[ctid].ContentTy

               }
            }
            //Return those that we can map
            return rt;
         }
         return base.LoadChildrenReferencesAndTypes(contentLink, languageID, out languageSpecific);
      }

      /// <summary>
      /// Loads content items
      /// </summary>
      /// <param name="contentLink"></param>
      /// <param name="languageSelector"></param>
      /// <returns></returns>
      protected override IContent LoadContent(ContentReference contentLink, ILanguageSelector languageSelector)
      {
         var lnk = IdentityMappingService.Get(contentLink);
         string entryid = lnk.ExternalIdentifier.AbsolutePath.TrimStart('/');
         //Future improvement: Handle languages, load linked content and assets as well
         var entry=_Client.GetEntry<dynamic>(entryid).Result;
         return GenerateContentFromEntry(entry);
      }

      /// <summary>
      /// Generates the IContent entries
      /// </summary>
      /// <param name="entry"></param>
      /// <returns></returns>
      protected IContent GenerateContentFromEntry(dynamic entry)
      {
         string ctid = entry.sys.contentType.sys.id;
         var map = TypeMapper.TypeMappings[ctid];
         string entryid = entry.sys.id;
         var id = IdentityMappingService.Get(new Uri(new Uri(BASEURI), entryid), true);

         ContentType type = TypeRepo.Load(map.ContentType); //Update
         var fc = _ContentFactory.CreateContent(type, new EPiServer.Construction.BuildingContext(type)
         {
            Parent = ContentRepo.Get<ContentFolder>(EntryPoint), ContentLink=id.ContentLink
         }) as IContent;

         fc.ContentGuid = id.ContentGuid;
         map.MapAllProperties(fc, entry);

         (fc as IVersionable).Status = VersionStatus.Published;
         (fc as IVersionable).IsPendingPublish = false;
         (fc as ILocalizable).Language = CultureInfo.GetCultureInfo("en");
         (fc as ILocalizable).MasterLanguage = CultureInfo.InvariantCulture;
         (fc as ILocalizable).ExistingLanguages = new List<CultureInfo>();
         (fc as IReadOnly).MakeReadOnly();
         return fc as IContent;
      }


      //Avoid caching for now. Future: Cache, but cancel caching based on webhooks
      protected override void SetCacheSettings(IContent content, CacheSettings cacheSettings)
      {
         base.SetCacheSettings(content, cacheSettings);
         cacheSettings.CancelCaching = true;
      }
      protected override void SetCacheSettings(ContentReference contentReference, IEnumerable<GetChildrenReferenceResult> children, CacheSettings cacheSet
      {
         base.SetCacheSettings(contentReference, children, cacheSettings);
         cacheSettings.CancelCaching = true;
      }
      protected override void SetCacheSettings(ContentReference parentLink, string urlSegment, IEnumerable<MatchingSegmentResult> childrenMatches, Cach
      {
         base.SetCacheSettings(parentLink, urlSegment, childrenMatches, cacheSettings);
         cacheSettings.CancelCaching = true;
      }

      protected override ContentResolveResult ResolveContent(ContentReference contentLink)
      {
         ContentResolveResult crr = new ContentResolveResult();
         crr.ContentLink = contentLink;
         var id=IdentityMappingService.Get(contentLink);
         if (id == null) return null;
         crr.UniqueID = id.ContentGuid;
         return crr;
      }

      protected override ContentResolveResult ResolveContent(Guid contentGuid)
      {
         ContentResolveResult crr = new ContentResolveResult();
         crr.UniqueID = contentGuid;
         var id = IdentityMappingService.Get(contentGuid);
         if (id == null) return null;
         crr.ContentLink = id.ContentLink;
         return crr;
      }

      public override void Initialize(string name, NameValueCollection config)
      {
         base.Initialize(name, config);
         var httpClient = new HttpClient();
         _Client = new ContentfulClient(httpClient, WebConfigurationManager.AppSettings["Contentful:ApiKey"], WebConfigurationManager.AppSettings["Conter

      }


   }
}
```

ContentfulContentProvider.cs hosted with ♥ by GitHub view raw

```
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
```

```cs
using System.Linq;
using ContentfulDemo.Contentful;
using ContentfulDemo.Models.Blocks;
using EPiServer;
using EPiServer.Configuration;
using EPiServer.Core;
using EPiServer.DataAccess;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.Security;
using EPiServer.ServiceLocation;

namespace ContentfulDemo.ContentfulTools
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class ContentfulInit : IInitializableModule
    {

        public void Initialize(InitializationEngine context)
        {

            //Setup mapping
            var mapper = ServiceLocator.Current.GetInstance<ExternalTypeMapper>();
            mapper.MapType<CFBlogBlock>("blogPost").Map((a, b) =>
                {
                    a.Name = Convert.ToString(b.title);
                    (a as CFBlogBlock).Description = Convert.ToString(b.description);
                    (a as CFBlogBlock).Body = Convert.ToString(b.body);
                } );


        }

        public void Uninitialize(InitializationEngine context)
        {
            //Add uninitialization logic
        }
    }
}
```

ContentfulInit.cs hosted with ♥ by GitHub view raw

```cs
using Contentful.Core.Models;
using EPiServer.Core;
using EPiServer.ServiceLocation;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Reflection;
using System.Web;

namespace ContentfulDemo.Contentful
{
    [ServiceConfiguration(Lifecycle = ServiceInstanceScope.Singleton, ServiceType = typeof(ExternalTypeMapper))]
    public class ExternalTypeMapper
    {
        public Dictionary<string, ExternalTypeMapping> TypeMappings { get; set; }


        public ExternalTypeMapping MapType<G>(string Identifier)
        {
            //Return a type mapping object?
            var rt = new ExternalTypeMapping();
            rt.Identifier = Identifier;
            rt.ContentType = typeof(G);
            TypeMappings.Add(Identifier, rt);
            return rt;
        }


        public ExternalTypeMapper()
        {
            TypeMappings = new Dictionary<string, ExternalTypeMapping>();
        }
    }



    public class ExternalTypeMapping
    {

        public Action<EPiServer.Core.IContent, dynamic> MapAllProperties { get; set; }

        public ExternalTypeMapping Map(Action<EPiServer.Core.IContent, dynamic> map)
        {
            this.MapAllProperties = map;
            return this;
        }

        public string Identifier { get; set; }
        public Type ContentType { get; set; }


    }
}
```

ExternalTypeMapper.cs hosted with ♥ by GitHub view raw

Vision Demos & Prototypes | Integrations | Optimizely (Episerver) | Contentful | CMS

RECENT POSTS

The Future Might be Distributed
Gist Content Provider
Content Provider or Replication

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