ALLAN THRAEN | 5 years ago | PDF |

Integrations Addon Development .NET Development CMS Optimizely (Episerver)

GIST CONTENT PROVIDER

Always preferring coding over 'real work' I figured that it would be pretty neat if I could just drag and drop my gists on GitHub directly into my blog posts here in Episerver in order to embed them. Naturally, a content provider seemed like the right choice...

Content Providers are awesome! I've had so much fun with them over the years - starting back from when they were named Page Providers (See for instance this archive post if you are feeling nostalgic - it's from August 2008: Building your own Page Provider: Northwind).

They can be very powerful when used right - but with great power comes great responsibility and there are many, many hidden man traps in the content provider jungle. I will touch on a few of them in this blog post.

If you are new to content providers, I strongly recommend first reading Per Magne's series from 2014, starting here then part 2 and finally part 3. There are many other good articles on content providers, but I feel like his is one of the best introductions.

Now, what I had in mind was basically just a folder in my block structure with gist blocks - automatically fetching them from GitHub. You should then be able to drag them to a content area or a XHTML field and the gist would get embedded. Simple, right? Read-only, using standard blocks, no versioning, no language handling, no sophisticated UI. GitHub is as always a great tool for developers, and of course they have a REST API for fetching Gists for a user. And it even allows anonymous access, so we don't even have to worry about OAuth for this call. It's well documented here: https://developer.github.com/v3/gists/#list-a-users-gists. Basically you can call https://api.github.com/users/[username]/gists to get a json list of public Gists. Go ahead and try (you know you want to).

First thing I did was to make a simple helper class to the Github API - it has a method that basically fetches the gists and returns them in an IEnumerable. It's pretty straight forward and you can see it below, in the embedded Gist (and yes, it is embedded using the Gist content provider). I also made a Gist Block type in Episerver - inheriting from a BlockData like any other block on your site and added a few of the values available in the Json feed.

But here comes the fun part - time to build the actual provider. Inherit from "ContentProvider" - and we are ready for some magic.
There will probably not be an unlimited amount of gists for a given user, so I figure they are fairly safe to load into memory - as opposed to calling GitHub constantly which could slow down everything significantly.
So I make a method called "GetGists()" which will return them - either from cache, or if they are not in cache it will load them and add them to cache before returning them. To make things even easier for myself I load them straight into GistBlocks ready to be returned.

```
/// <summary>
/// Get Gists from Cache
/// </summary>
/// <returns></returns>
protected List<GistBlock> GetGists()
                                    {
    var cache = ServiceLocator.Current.GetInstance<ISynchronizedObjectInstanceCache>();
    var rt = cache.Get<List<GistBlock>>(KEY, ReadStrategy.Immediate);
    if (rt == null)
    {
        rt = LoadGists();
        cache.Insert(KEY, rt, new CacheEvictionPolicy(new TimeSpan(0, 30, 0), CacheTimeoutTyp
    }
    return rt;
}

/// <summary>
/// Load Gists from Github and create objects to put in cache
/// </summary>
/// <returns></returns>
private List<GistBlock> LoadGists()
                                   {
    //Load Gists
    var gists = GistHelper.LoadGists(Username);
    var _typeRepo = ServiceLocator.Current.GetInstance<IContentTypeRepository>();
    var _contentFactory = ServiceLocator.Current.GetInstance<IContentFactory>();
    var _contentRepo = ServiceLocator.Current.GetInstance<IContentRepository>();
    ContentType type = _typeRepo.Load(typeof(GistBlock));
    int i = 1000;
    var Gists = new List<GistBlock>(gists.Count());
    foreach (var g in gists.OrderBy(g => g.Created))
    {
        var fc = _contentFactory.CreateContent(type, new EPiServer.Construction.BuildingConte
        {
            Parent = _contentRepo.Get<ContentFolder>(EntryPoint)
        }) as GistBlock;
        fc.Code = g.Id;
        (fc as IContent).Name = g.Files.First();
        fc.Description = g.Description;
```

```
                fc.HtmlUrl = new Url(g.HtmlUrl);
                fc.User = Username;
                (fc as IVersionable).Status = VersionStatus.Published;
                (fc as IVersionable).IsPendingPublish = false;
                (fc as IVersionable).StartPublish = DateTime.Now;
                (fc as ILocalizable).Language = CultureInfo.GetCultureInfo("en");
                (fc as IContent).ContentLink = new ContentReference(i, this.ProviderKey);
                (fc as IContent).ContentGuid = GuidFromId(i);
                (fc as IChangeTrackable).Changed = g.Modified;
                (fc as IChangeTrackable).CreatedBy = Username;
                (fc as IChangeTrackable).Created = g.Created;
                (fc as ILocalizable).MasterLanguage = CultureInfo.InvariantCulture;
                (fc as ILocalizable).ExistingLanguages = new List<CultureInfo>();
                fc.MakeReadOnly();
                Gists.Add(fc);
                i++;
            }
            return Gists;
        }
```

Here is a little caveat. Depending on what kind of IContent you decide to return from your content provider, different properties might have to be set - and it can be hard to know which ones are mandatory. Sometimes a property that you forget to set, results in a hanging UI or "server is offline" messages. Remember that IContent can be pages, blocks, media, products, content folders, or any other type you decide to create that just implements IContent. In my case I'm inheriting a regular block that inherits BlockData and that comes with both benefits and obligations. Cause blocks are for instance both versioned and multi-language, so even if your provider is not you still have to take that into account. For example, forgetting to set MasterLanguage as I did initially will result in a slightly weird and hidden null-reference exception that took some decompiling of Episerver source code to figure out.

Another, tricky part is ID/Guid/Url resolving. Most of the time Episerver uses ContentReferences to locate content - and they have an integer ID. But all content should also have a unique Guid which is primarily used for permanent links - permanent like the ones that are stored when you add a block to a content area or in a XHTML field. And a content provider is expected to:
a) Be able to know if a Guid or an ID belongs to it's content
b) Be able to map between Guids and IDs.
Urls' is a slightly different story, but not that necessary here as blocks by definition not are url adressable.
If you forget to handle that mapping you'll see some weird behavior. For instance that you can add your content to a content area in the UI, but when you refresh the page or try to publish it will magically disappear. And it's pretty hard to debug if you don't know about the need to resolve guids and ids.
This is not a new problem. Back in the time of Page Providers I made a 'MappedPageProvider' to solve it. It would map an external string key to ID's and Guids. These days that problem is now solved by the IdentityMappingService as Per Magne shows in his posts. But from what I can read in forums the performance of that is sometimes cause for problems - and I figured, why not avoid it if possible :-)

My solution is to map the 4 bytes an Int32 ID into the first 4 bytes of a 32 byte Guid :-) Maybe not the prettiest solution in the world, but it works.

```
        public static Guid BASEGUID = new Guid("2900353B-DFFF-4EB8-A9BF-3CE237EFA96F");

        public static int IdFromGuid(Guid g)
                                            {
            var b = BitConverter.ToInt32(g.ToByteArray().Take(4).ToArray(), 0);
            return b;
        }

        public static bool IsGuidOk(Guid g)
                                           {
            var b1 = BASEGUID.ToByteArray();
            var b2 = g.ToByteArray();
            for (int i = 4; i < b1.Length; i++)
                if (b1[i] != b2[i]) return false;
            return true;
        }

        public static Guid GuidFromId(int Id)
                                             {
            var b1 = BASEGUID.ToByteArray();
            var b2 = BitConverter.GetBytes(Id);
            for (int i = 0; i < b2.Length; i++) b1[i] = b2[i];
            return new Guid(b1);
        }
```

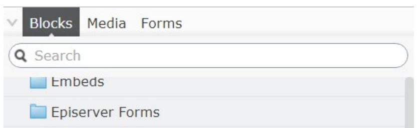
With this in place it's not hard to build the required Resolve methods:

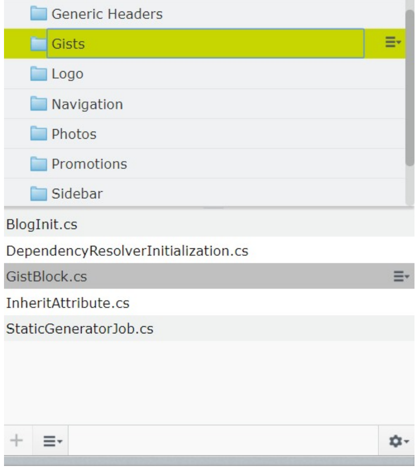
```
        protected override ContentResolveResult ResolveContent(Guid contentGuid)
                                                                                {
            if (!IsGuidOk(contentGuid)) return null; //Not ours
            ContentResolveResult crr = new ContentResolveResult();
            crr.ContentLink = new ContentReference(IdFromGuid(contentGuid),this.ProviderKey);
            var content = LoadContent(crr.ContentLink, null);
            crr.UniqueID = contentGuid;
            crr.ContentUri = ConstructContentUri(content.ContentTypeID, crr.ContentLink, crr.UniqueID)
            return crr;
        }
```

The rest of the content provider itself is pretty straight forward - so I'm not going to dive too deep into that. The 'important' methods are of course the LoadContent, for loading the actual content, and the GetChildrenReferencesAndTypes for loading the hierarchy (which is easily done with a flat structure as I have in this case).

There are many other bits to this circus. Like the initialization that registers the provider - a nice alternative to doing it in the web.config.
A side note here: I have an idea for a completely new approach to attaching content providers, but that will have to wait for another blog post.
I've also included a GistBlockController that outputs the needed embedding code as well as a UIDescriptor that turns off preview/on-page-edit views for the Gists, so I don't have to worry about those for now.

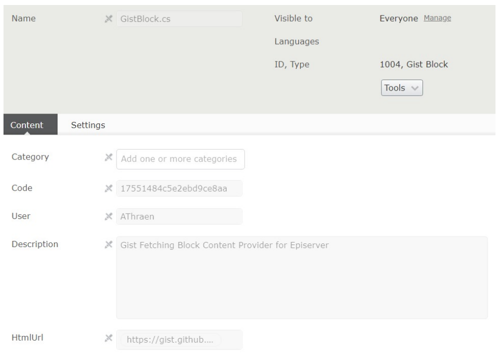
The end result? This is what it looks like:





(Note that for name, I've picked the first filename - as you can't be certain of a description text that looks like a name).

This is what it looks like if you open a gist in all properties view:



Name: GistBlock.cs
Visible to: Everyone Manage
Languages
ID, Type: 1004, Gist Block

Category: Add one or more categories
Code: 17551484c5e2ebd9ce8aa
User: AThraen
Description: Gist Fetching Block Content Provider for Episerver
HtmlUrl: https://gist.github....

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

namespace AllanTech.Web.Business.ContentProviders
{
    /// <summary>
    /// Define the Block type to hold the Gist
    /// </summary>
    [ContentType(DisplayName = "Gist Block", GUID = "bca5e15d-976a-4b0e-8dc0-2b165e052170", Description = "", AvailableInEditMode =false)]
    public class GistBlock : BlockData
    {
        [Editable(false)]
        public virtual string Code { get; set; }

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

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

        [Editable(false)]
        public virtual Url HtmlUrl { get; set; }
    }
}
```

GistBlock.cs hosted with ❤ by GitHub view raw

```
using System.Web.Mvc;
using EPiServer.Web.Mvc;

namespace AllanTech.Web.Business.ContentProviders
{
    /// <summary>
    /// Display the Gist Block
    /// </summary>
    public class GistBlockController : BlockController<GistBlock>
    {
        public const string SCRIPTLINE = "<script src=\"https://gist.github.com/{0}/{1}.js\"></script>";

        public override ActionResult Index(GistBlock currentBlock)
        {
            return Content(string.Format(SCRIPTLINE, currentBlock.User, currentBlock.Code));
        }
    }
}
```

GistBlockController.cs hosted with ❤ by GitHub view raw

```
using EPiServer;
using EPiServer.Construction;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.Framework.Cache;
using EPiServer.ServiceLocation;
using EPiServer.Web;
using System;
using System.Collections.Generic;
using System.Collections.Specialized;
using System.Globalization;
using System.Linq;
using System.Web;

namespace AllanTech.Web.Business.ContentProviders
{
```

```
public class GistsContentProvider : ContentProvider
{
    public const string KEY = "GIST";

    public string Username { get; set; }


    //The following region maps from ID to Guids using the first 4 bytes of each guid to store the Int ID.
    #region GuidMapping
    public static Guid BASEGUID = new Guid("2900353B-DFFF-4EB8-A9BF-3CE237EFA96F");

    public static int IdFromGuid(Guid g)
    {
        var b = BitConverter.ToInt32(g.ToByteArray().Take(4).ToArray(), 0);
        return b;
    }

    public static bool IsGuidOk(Guid g)
    {
        var b1 = BASEGUID.ToByteArray();
        var b2 = g.ToByteArray();
        for (int i = 4; i < b1.Length; i++)
            if (b1[i] != b2[i]) return false;
        return true;
    }

    public static Guid GuidFromId(int Id)
    {
        var b1 = BASEGUID.ToByteArray();
        var b2 = BitConverter.GetBytes(Id);
        for (int i = 0; i < b2.Length; i++) b1[i] = b2[i];
        return new Guid(b1);
    }
    #endregion

    /// <summary>
    /// Return a gistblock
    /// </summary>
    /// <param name="contentLink"></param>
    /// <param name="languageSelector"></param>
    /// <returns></returns>
    protected override IContent LoadContent(ContentReference contentLink, ILanguageSelector languageSelector)
    {
        var gist=GetGists().Where(gb => (gb as IContent).ContentLink.CompareToIgnoreWorkID(contentLink)).Cast<IContent>().FirstOrDefault();
        return gist;
    }

    /// <summary>
    /// Used to fetch tree structure
    /// </summary>
    /// <param name="contentLink"></param>
    /// <param name="languageID"></param>
    /// <param name="languageSpecific"></param>
    /// <returns></returns>
    protected override IList<GetChildrenReferenceResult> LoadChildrenReferencesAndTypes(ContentReference contentLink, string languageID, out bool languag
    {
        languageSpecific = false;
        if (contentLink.CompareToIgnoreWorkID(EntryPoint))
        {
            //Root, list items
            return GetGists().Select(g => new GetChildrenReferenceResult() { ContentLink = (g as IContent).ContentLink, IsLeafNode = true, ModelType = typeof(GistE
        }
        return base.LoadChildrenReferencesAndTypes(contentLink, languageID, out languageSpecific);
    }

    /// <summary>
    /// Used to resolve Guid, IDs and urls. Needs to work for Gists to remain in content areas.
    /// </summary>
    /// <param name="contentLink"></param>
    /// <returns></returns>
    protected override ContentResolveResult ResolveContent(ContentReference contentLink)
    {
        //Tricky bits
        if (contentLink.ProviderName != this.ProviderKey) return null; //Not ours
        ContentResolveResult crr = new ContentResolveResult();
        crr.ContentLink = contentLink;
        var content = LoadContent(contentLink, null);
        crr.UniqueID = content.ContentGuid;
        crr.ContentUri = ConstructContentUri(content.ContentTypeID, crr.ContentLink, crr.UniqueID);
        return crr;
    }

    protected override ContentResolveResult ResolveContent(Guid contentGuid)
    {
        if (!IsGuidOk(contentGuid)) return null; //Not ours
        ContentResolveResult crr = new ContentResolveResult();
        crr.ContentLink = new ContentReference(IdFromGuid(contentGuid),this.ProviderKey);
        var content = LoadContent(crr.ContentLink, null);
        crr.UniqueID = contentGuid;
        crr.ContentUri = ConstructContentUri(content.ContentTypeID, crr.ContentLink, crr.UniqueID);
        return crr;
    }

    public override void Initialize(string name, NameValueCollection config)
    {
        base.Initialize(name, config);
        if (config["username"] != null) Username = config["username"];
        LoadGists();
    }

    /// <summary>
    /// Get Gists from Cache
    /// </summary>
    /// <returns></returns>
    protected List<GistBlock> GetGists()
    {
        var cache = ServiceLocator.Current.GetInstance<ISynchronizedObjectInstanceCache>();
        var rt = cache.Get<List<GistBlock>>(KEY, ReadStrategy.Immediate);
        if (rt == null)
        {
            rt = LoadGists();
            cache.Insert(KEY, rt, new CacheEvictionPolicy(new TimeSpan(0, 30, 0), CacheTimeoutType.Absolute));
        }
        return rt;
    }

    /// <summary>
    /// Load Gists from Github and create objects to put in cache
    /// </summary>
    /// <returns></returns>
    private List<GistBlock> LoadGists()
    {
        //Load Gists
```

```csharp
            var gists = GistHelper.LoadGists(Username);
            var _typeRepo = ServiceLocator.Current.GetInstance<IContentTypeRepository>();
            var _contentFactory = ServiceLocator.Current.GetInstance<IContentFactory>();
            var _contentRepo = ServiceLocator.Current.GetInstance<IContentRepository>();
            ContentType type = _typeRepo.Load(typeof(GistBlock));
            int i = 1000;
            var Gists = new List<GistBlock>(gists.Count());
            foreach (var g in gists.OrderBy(g => g.Created))
            {
                var fc = _contentFactory.CreateContent(type, new EPiServer.Construction.BuildingContext(type)
                {
                    Parent = _contentRepo.Get<ContentFolder>(EntryPoint)
                }) as GistBlock;
                fc.Code = g.Id;
                (fc as IContent).Name = g.Files.First();
                fc.Description = g.Description;
                fc.HtmlUrl = new Url(g.HtmlUrl);
                fc.User = Username;
                (fc as IVersionable).Status = VersionStatus.Published;
                (fc as IVersionable).IsPendingPublish = false;
                (fc as IVersionable).StartPublish = DateTime.Now;
                (fc as ILocalizable).Language = CultureInfo.GetCultureInfo("en");
                (fc as IContent).ContentLink = new ContentReference(i, this.ProviderKey);
                (fc as IContent).ContentGuid = GuidFromId(i);
                (fc as IChangeTrackable).Changed = g.Modified;
                (fc as IChangeTrackable).CreatedBy = Username;
                (fc as IChangeTrackable).Created = g.Created;
                (fc as ILocalizable).MasterLanguage = CultureInfo.InvariantCulture;
                (fc as ILocalizable).ExistingLanguages = new List<CultureInfo>();
                fc.MakeReadOnly();
                Gists.Add(fc);
                i++;
            }
            return Gists;
        }
    }
}
```

GistContentProvider.cs hosted with ❤ by GitHub view raw

```csharp
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web;

namespace AllanTech.Web.Business.ContentProviders
{
    /// <summary>
    /// Helper class to fetch the gists from github
    /// </summary>
    public class GistHelper
    {
        public const string APIPath = "https://api.github.com/users/{0}/gists";

        public static IEnumerable<Gist> LoadGists(string Username)
        {
            //Load url, create gist objects and return them
            WebClient wc = new WebClient();
            wc.Headers.Add(HttpRequestHeader.UserAgent, "Custom");
            string s = wc.DownloadString(string.Format(APIPath, Username));
            var array = JArray.Parse(s);
            foreach (var gist in array)
            {
                var g = new Gist();
                g.Description = gist["description"].Value<string>();
                g.Id = gist["id"].Value<string>();
                g.HtmlUrl = gist["html_url"].Value<string>();
                g.Created = gist["created_at"].Value<DateTime>();
                g.Modified = gist["updated_at"].Value<DateTime>();
                var files = gist["files"].Values();
                g.Files = files.Select(j => j["filename"].Value<string>()).ToList();
                yield return g;
            }
        }
    }

    /// <summary>
    /// The Gist as it is returned from Github
    /// </summary>
    public class Gist
    {
        public string Id { get; set; }
        public string Description { get; set; }
        public string HtmlUrl { get; set; }
        public DateTime Created { get; set; }
        public DateTime Modified { get; set; }
        public List<string> Files { get; set; }
    }
}
```

GistHelper.cs hosted with ❤ by GitHub view raw

```csharp
using System;
using System.Collections.Specialized;
using System.Linq;
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 AllanTech.Web.Business.ContentProviders
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class GistProviderInitializer : IInitializableModule
    {
        public static ContentFolder GetEntryPoint(string name)
        {
            var contentRepository = ServiceLocator.Current.GetInstance<IContentRepository>();
            var folder = contentRepository.GetBySegment(ContentReference.GlobalBlockFolder, name, LanguageSelector.AutoDetect()) as ContentFolder;
            if (folder == null)
            {
                folder = contentRepository.GetDefault<ContentFolder>(ContentReference.GlobalBlockFolder);
                folder.Name = name;
                contentRepository.Save(folder, SaveAction.Publish, AccessLevel.NoAccess);
            }
            return folder;
        }
        /// <summary>
        /// Alternative to defining in web.config.
        /// </summary>
        /// <param name="context"></param>
        public void Initialize(InitializationEngine context)
        {
            var gistprovider = new GistsContentProvider();

            var providerValues = new NameValueCollection();
            var entrypoint = GetEntryPoint("Gists").ContentLink;
            providerValues.Add(ContentProviderElement.EntryPointString, entrypoint.ToString());
            providerValues.Add(ContentProviderElement.CapabilitiesString, "None");
            providerValues.Add("Username", "AThraen");

            gistprovider.Initialize(GistsContentProvider.KEY, providerValues);
            var providerManager = context.Locate.Advanced.GetInstance<IContentProviderManager>();
            providerManager.ProviderMap.AddProvider(gistprovider);
        }

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

GistProviderInitializer.cs hosted with ❤ by GitHub view raw

```csharp
using EPiServer.Shell;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace AllanTech.Web.Business.ContentProviders
{
    [UIDescriptorRegistration]
    public class GistUIDescriptor : UIDescriptor<GistBlock>
    {
        public GistUIDescriptor() : base("icon-document")
        {
            DefaultView = CmsViewNames.AllPropertiesView;
            this.DisabledViews = new List<string>();
            this.DisabledViews.Add(CmsViewNames.OnPageEditView);
            this.DisabledViews.Add(CmsViewNames.PreviewView);
        }
    }
}
```

GistUIDescriptor.cs hosted with ❤ by GitHub view raw

Integrations Addon Development .NET Development CMS Optimizely (Episerver)

RECENT POSTS

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