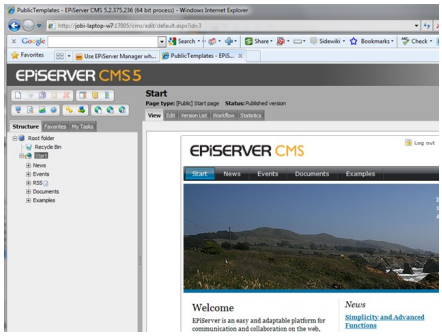
ALLAN THRAEN | 6 years ago | PDF |

C# Optimizely (Episerver)

GOOD OL' DYNAMIC PROPERTIES



There was a time, when men were made of steel, ships made of wood, Episerver was spelled with a weird capitalization and the CMS had something called Dynamic Properties that was usually misused. They've been gone for a while, but I miss them, so here's yet another attempt at solving the property inheritance challenge.

To those of you, my dear readers, that have no clue what Dynamic Properties were all about, let me begin with enlightening you:

Once upon a time, there was a mythical feature, loved by some and hated by many called Dynamic Propeties. Dynamic Properties were properties that were not set on the content itself, but rather could be inherited across all content types, throughout the content hierachy. A good example would be a dynamic property called something like "SecondLevelMenuRoot" that would be used to know which child objects should be listed in the sidebar menu. Usually, a super-user editor that could find his/her way into the secret edit menu for dynamic properties would then set the property to the each of the first level items. That way, the property would inherit down and all leaf nodes would show the second level menu corresponding to their place. And the world was a better place.

Sadly, evil forced tended to abuse these great powers and often ended up making many, many dynamic properties and use them for stuff like a LogoImageLink or SearchButtonText that really should have been a site setting instead. Since Dynamic properties had to be resolved dynamically (hence the name) that tended to slow down the sites quite a bit.

But, surely we are smarter now and ready to once again unleash this power, right?

In any case, I needed some inheritance badly when building this blog. Why?

Well, here is a good use-case: On each blog post I have a sidebar content area. Usually I'll have the same blocks there on all blocks - but there'll probably one or two where I'll want something different placed there. I don't want to have to remember to put the same blocks in there for every single blog post. And I don't want it as 'Default' value upon creation, cause what happens when I decide to change the blocks?

I could perhaps make 1 giant Block to rule all blocks, that would contain another content area with the other blocks....But blocks-in-blocks are kind of ugly in my eyes and should be avoided whenever possible (yes I know, I've done blocks in blocks both with Forms and Self Optimizing Block - but those were exceptions, ok).

Enough talk, let's see some code. I decided that one of the best approaches would be if we could let the developers decide which properties should be inherited and how it should work. Something like this:

```
//Sidebar
[Inherit(InheritIfNullOrEmpty =true, ParentPropertyToInheritFrom ="DefaultChildSideBar", SearchAllAnc
public virtual ContentArea SideBar { get; set; }
```

What's happening here is basically just that on my Blog Post content type, I'm telling it that this property value should be inherited, if it's not set (null or empty). It should try to inherit from the parent page, if the parent page has a property called "DefaultChildSideBar". If I hadn't specified that, it would simply look for a parent property of the same name as the current property. I also tell it to search all ancestors - so if the parent doesn't have that property set, it should look to the grandparent and so on.

Sometimes you might like to let the editor decide if a value should be inherited or not - in that case, you could add another boolean property on the page and specify it's name as "SwitchPropertyName" in the attribute.

I haven't yet decided on a good strategy for when to populate the properties - so for now, I've let it be up to the developers - they basically have to call an extension method on IContent that will attempt to populate the needed inherited properties.

```
public static T PopulateInheritedProperties<T>(this T Content) where T : PageData
        {
            var rt = (Content as IReadOnly).CreateWritableClone() as PageData;
            var props = Content.GetPropertiesWithAttribute(typeof(InheritAttribute));
            bool modified = false;
            foreach (var prop in props)
            {
                var attr = prop.GetCustomAttribute<InheritAttribute>(true);

                if (
                    (!String.IsNullOrEmpty(attr.SwitchPropertyName) && ((bool)Content.GetType().GetPr
                    ((attr.InheritIfNull || attr.InheritIfNullOrEmpty) && (prop.GetValue(Content) ==
                    (attr.InheritIfNullOrEmpty && ((prop.PropertyType == typeof(ContentArea)) && (prop
                    )
                {
                    //Resolve Inherited Properties
                    var repo = ServiceLocator.Current.GetInstance<IContentRepository>();
                    foreach(var a in repo.GetAncestors(Content.ContentLink).Take((attr.SearchAllAncest
                    {
```

```
                    var parentprop = (a as IContentData).Property[attr.ParentPropertyToInheritFro
                    if (parentprop!=null && !parentprop.IsNull)
                    {
                        prop.SetValue(rt, parentprop.Value);
                        modified = true;
                        break;
                    }
                }
            }
        }
        if (modified)
        {
            rt.MakeReadOnly();
            return rt as T;
        }
        return Content;

    }
```

The attribute presents these options:

```
public class InheritAttribute : Attribute
{
    /// <summary>
    /// Name of Boolean property that indicates if this property should be inherited
    /// </summary>
    public string SwitchPropertyName { get; set; }

    /// <summary>
    /// Inherit this value if it's null
    /// </summary>
    public bool InheritIfNull { get; set; }

    /// <summary>
    /// Inherit this value if it's null or empty
    /// </summary>
    public bool InheritIfNullOrEmpty { get; set; }

    /// <summary>
    /// Name of property on parent content to inherit from. Default is same name.
    /// </summary>
    public string ParentPropertyToInheritFrom { get; set; }

    /// <summary>
    /// Keep searching ancestors until Root
    /// </summary>
    public bool SearchAllAncestors { get; set; }

}
```

You can see the entire Gist below.

A word of caution

This is in no way a done solution - in fact, it's a very first draft. I just figured I'd share it here to get some feedback (which is very welcome in the comments below).

There are still many pieces to the puzzle missing. For example:

- What about Blocks? What will they inherit from?
- Should we also try to resolve inheritance recursively on parents with inherited properties?
- When should we resolve? Currently I resolve inherited properties by calling the method in my Controller.
- How can we alter the UI to make the editor aware that a certain property is begin inherited - and from where it's being inherited?

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace AllanTech.Web.PropertyInheritance
{
  public class InheritAttribute : Attribute
  {
    /// <summary>
    /// Name of Boolean property that indicates if this property should be inherited
    /// </summary>
    public string SwitchPropertyName { get; set; }

    /// <summary>
    /// Inherit this value if it's null
    /// </summary>
    public bool InheritIfNull { get; set; }

    /// <summary>
    /// Inherit this value if it's null or empty
    /// </summary>
    public bool InheritIfNullOrEmpty { get; set; }

    /// <summary>
    /// Name of property on parent content to inherit from. Default is same name.
    /// </summary>
    public string ParentPropertyToInheritFrom { get; set; }

    /// <summary>
    /// Keep searching ancestors until Root
    /// </summary>
    public bool SearchAllAncestors { get; set; }

  }
}
```

InheritAttribute.cs hosted with ♡ by GitHub view raw

```
using EPiServer.Core;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```cs
using AllanTech.Web.Helpers;
using EPiServer.ServiceLocation;
using EPiServer;
using EPiServer.Data.Entity;
using System.Reflection;

namespace AllanTech.Web.PropertyInheritance
{
    public static class PropertyInheritor
    {

        public static T PopulateInheritedProperties<T>(this T Content) where T : PageData
        {
            var rt = (Content as IReadOnly).CreateWritableClone() as PageData;
            var props = Content.GetPropertiesWithAttribute(typeof(InheritAttribute));
            bool modified = false;
            foreach (var prop in props)
            {
                var attr = prop.GetCustomAttribute<InheritAttribute>(true);

                if (
                    (!String.IsNullOrEmpty(attr.SwitchPropertyName) && ((bool)Content.GetType().GetProperty(attr.SwitchPropertyName).GetValue(Content))) ||
                    ((attr.InheritIfNull || attr.InheritIfNullOrEmpty) && (prop.GetValue(Content) == null)) ||
                    (attr.InheritIfNullOrEmpty && ((prop.PropertyType == typeof(ContentArea)) && (prop.GetValue(Content) as ContentArea).Count == 0))
                    )
                {
                    //Resolve Inherited Properties
                    var repo = ServiceLocator.Current.GetInstance<IContentRepository>();
                    foreach(var a in repo.GetAncestors(Content.ContentLink).Take((attr.SearchAllAncestors)?1000:1))
                    {
                        var parentprop = (a as IContentData).Property[attr.ParentPropertyToInheritFrom ?? prop.Name];
                        if (parentprop!=null && !parentprop.IsNull)
                        {
                            prop.SetValue(rt, parentprop.Value);
                            modified = true;
                            break;
                        }
                    }
                }
            }
            if (modified)
            {
                rt.MakeReadOnly();
                return rt as T;
            }
            return Content;

        }


    }
}
```

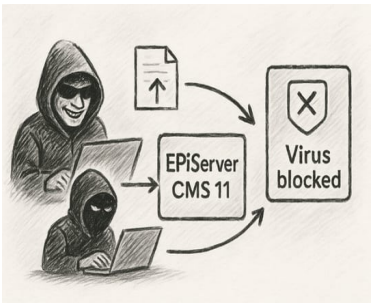
PropertyInheritor.cs hosted with ♡ by GitHub view raw

ADDING GRANULAR EDITOR ACCESS CONTROL TO BUILT-IN PARTS OF OPTIMIZELY CMS 12

A classic challenge in Optimizely CMS (well, really in any system I guess), is to ensure that the right people have the right access - and that potentially dangerous actions can't be accidentally done by unqualified users.

Optimizely (Episerver) .NET Development CMS Tips and Tricks C#

May 30 2025

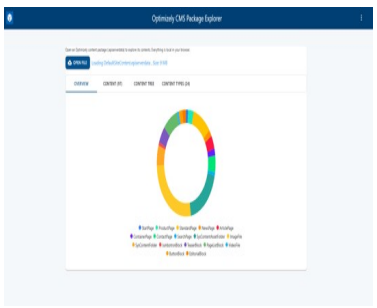


SCAN FILE UPLOADS FOR MALWARE IN EPISERVER/OPTIMIZELY CMS 11 - EPISERVER FORMS

Do you have forms on your website where visitors can upload files? Perhaps CV's for job applications or documentation for claims, or other kind of applications or images? And have you thought about the risk of these files potentially containing malware right on your production webserver? A client of mine has this concern on an EPiServer (now Optimizely) CMS 11 using the EPiServer Forms extension and I investigated and found an approach to handle it.

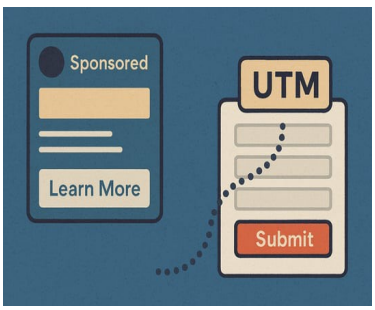
.NET Development Optimizely (Episerver) C# Tips and Tricks

May 9 2025



OPENSOURCE RELEASE: NEW PACKAGE EXPLORER FOR OPTIMIZELY CMS

The import/export ".episerverdata" packages have been around as far as I can remember - and even though they might seem a bit outdated, it's still one of the most common ways to move content around today. But it can be quite a hassle to work with. I recently was inspired to build a tool that will hopefully make life a bit easier when dealing with the packages



TRACKING UTM PARAMETERS IN OPTIMIZELY CMS FORMS

When you are spending your marketing dollars on social media / CPC campaigns, correctly attributing your leads is everything so you know where to invest more. Usually you can get this insight from your Marketing Automation or analytics - but I recently got a question if it's possible to also automatically add it to your Optimizely CMS forms. And of course it is. Here

across versions.

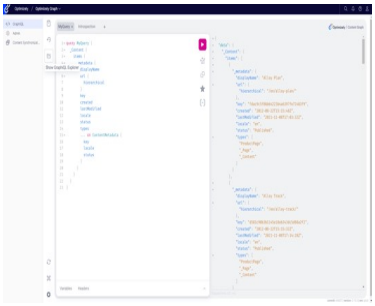
Vision Demos & Prototypes | Addon Development | Optimizely (Episerver)

April 9 2025

are two ways of doing that.

Optimizely (Episerver) | CMS | Website Improvements | Tips and Tricks | C#

April 3 2025

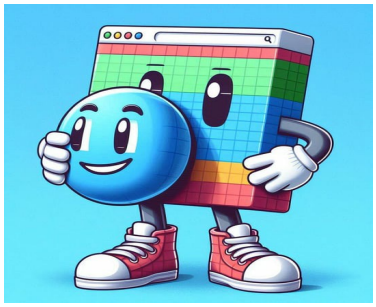


[SERIES: HEADLESS SITE ON OPTIMIZELY GRAPH] EXPLORING INTERESTING WAYS WE CAN QUERY THE CONTENT

Look up in the sky! It's a bird! No, wait - it's Optimizely Graph! This is part 2 in the blog post series where we explore how a fully functioning headless site can be build in .NET core using Optimizely Graph, and in this post we'll see how Optimizely Graph is both a powerful search & query engine (on par with good old Episerver Find) - but also how it can fully replace the content delivery API.

.NET Development | CMS | Optimizely (Episerver) | Information Retrieval | API Building

August 24 2024

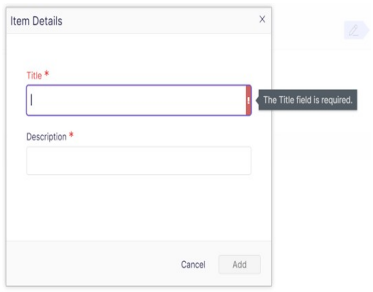


NEW SERIES: BUILDING A .NET CORE HEADLESS SITE ON OPTIMIZELY GRAPH AND SAAS CMS

Welcome to this new multi-post series where you can follow along as I indulge in yet another crazy experiment: Can we make our beloved Alloy site run as a headless site on top of Optimizely Graph (and SaaS CMS) using a mix of both MVC, Razor Pages and Blazor components - and can we make both the developer and editor experience as we are used to in classic Optimizely/Episerver sites? Let's find out!

CMS | .NET Development | Optimizely (Episerver)

June 14 2024



UPGRADE TO OPTIMIZELY CMS 12 ISSUE: LIST ITEM FIELDS HAVE BECOME REQUIRED

There are many funny details to be aware of when upgrading from Episerver CMS 11 to Optimizely CMS 12. One of them that might feel a bit confusing is when your list items start to have all required fields in the UI. Here's how to fix it.

C# | Optimizely (Episerver)

June 7 2024

CHRISTMAS COUNTDOWN: #1 THE GRAND FINALE. GOING HEADLESS WITHOUT USING YOUR HEAD!

In 2014 the term 'headless cms' was coined - and presented as a cool new 'feature' in the Web CMS industry. And it quickly became a hot buzzword for a few years later. In a few days it's 2024 and we can celebrate that it's been a concept for 10 years. Strangely, I still encounter new implementations that want to go 'headless' based on an almost religious belief that it's the new cool thing.

CMS | Optimizely (Episerver) | Frontend Development | Website Improvements | Tech Talk

December 23 2023

CHRISTMAS COUNTDOWN: #2 WE'RE LIVE! THAT MEANS WE'RE DONE, RIGHT?

They day you go live with your new website is naturally the culmination of months, sometimes years of work - and it's fine to celebrate. But #2 on this top 12 list of common pitfalls is to think that going live is the completion of the website. It's not. It's the start...

CMS | Optimizely (Episerver) | Website Improvements | Tech Talk

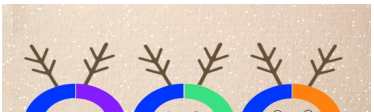
December 22 2023

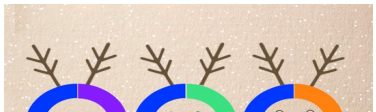
CHRISTMAS COUNTDOWN: #3 NIHS - NOT INVENTED HERE SYNDROME IN REAL LIFE

One of the most common and dreaded diseases in web site development often go undiagnosed and untreated for a long time. But it really should be, cause the effects are scary. Yes, I'm talking about the Not-Invented-Here Syndrome

.NET Development | Optimizely (Episerver) | CMS | Website Improvements | Tips and Tricks | Tech Talk

December 21 2023





CHRISTMAS COUNTDOWN: #4 EDITORS? IT'S JUST JOHN AND JANE, THEY KNOW ALL THE QUIRKS - WHY DOES EDIT-MODE MATTER?

An audience that is often neglected are the editors / content creators. That is a shame because happy editors => efficient editors => good content => great online experience.

CMS Optimizely (Episerver) Website Improvements Tips and Tricks Tech Talk

December 20 2023

CHRISTMAS COUNTDOWN: #5 SURE, OUR SERVERS ARE LOCKED UP TIGHT IN THE BASEMENT!

Securing your website is as important a topic as it is large and complex. In this post I will not go into too many details, but highlight a few problems I often see in Optimizely/EPiServer CMS implementations.

.NET Development CMS Optimizely (Episerver)

December 19 2023

‹ 1 2 3 4 5 6 7 8 9 10 11 ›

C# Optimizely (Episerver)

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

Copyright © 2025