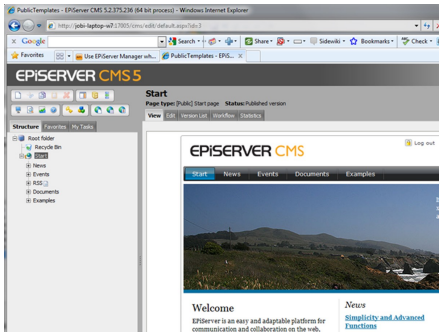
ALLAN THRAEN | 5 years ago | PDF |

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)) && (pro
                    )
                {
                    //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;
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

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

CHRISTMAS COUNTDOWN: #6 "WE LOVE CONTENT MODELS - WE HAVE _ALL_ OF THEM!"

The above statement is almost as scary as this: "Content Modelling - is that really needed? We just have one!"

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

December 18 2023

CHRISTMAS COUNTDOWN: #7 DDOS? WHAT'S THAT? WHAT DO YOU MEAN 'PREPARED'?

Is your website ready to handle intense usage scenarios like DDoS attacks or black friday? Many people think that testing performance is the same as testing for load - but it's not and sometimes it might even work against each other.

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

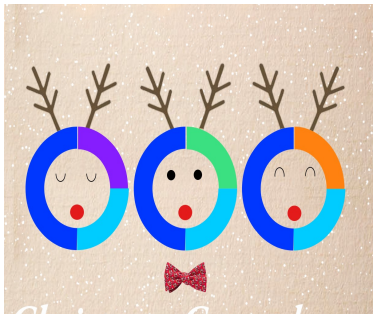
December 17 2023

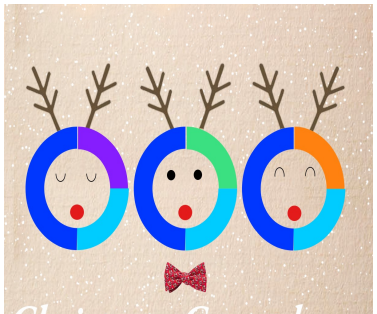
CHRISTMAS COUNTDOWN: #8 CODE MAINTENANCE IS 90% OF THE WORK

Greenfield development is by far the most fun for everybody. So it's easy to forget that most development work is actually maintenance. And every new line of code you write means more code to maintain. Almost all codebases I review have significant technical debt. And the debt starts to accumulate from the moment you start coding.

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

December 16 2023





CHRISTMAS COUNTDOWN: #9 WHAT? VIEWMODELS? NAH, WE DON'T NEED THOSE

This is another classic - with a big impact! Since recycling is great, why don't we just reuse the content model as a view model? We can just enrich it in the controller, right?

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

December 15 2023

CHRISTMAS COUNTDOWN: #10 IF IT'S OUT THERE, GOOGLE WILL EVENTUALLY FIND IT

Have you ever forgotten to protect stuff that wasn't meant to be public? If no, then you are probably a better person than me and most others - both developers and editors alike.

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

December 14 2023

CHRISTMAS COUNTDOWN: #11 DEPENDENCY INJECTION IS NOT AS EASY AS IT SEEMS

Dependency Injection is an extremely useful pattern. It has been used with EPiServer CMS for years - and with .NET Core it has truly become the go-to method of coupling your business logic together. However, once you start having services depend on other services their lifetimes can give some unexpected difficulties.

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

December 13 2023

CHRISTMAS COUNTDOWN: COMMON OPTIMIZELY CMS PITFALLS - #12 PICKING THE RIGHT ADD-ONS

12 days to Christmas and here is my countdown list of the top 12 common pitfalls I see in Optimizely CMS implementations - along with some tips on how to avoid them. Today we'll take a look at #12 on the list: Picking the right add-ons

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

December 12 2023

‹ 1 2 3 4 5 6 7 8 9 10 ›

C# Optimizely (Episerver)

RECENT POSTS

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