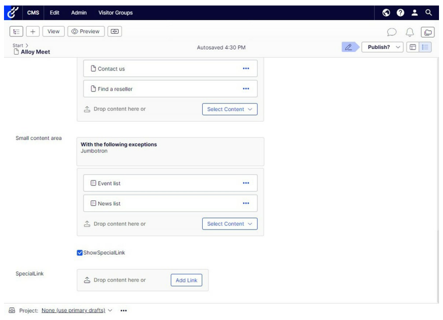
ALLAN THRAEN | 1 years ago | PDF |

C# Tips and Tricks Optimizely (Episerver)

PROPERTY DEPENDENCY IN OPTIMIZELY (EPISERVER) CMS 12

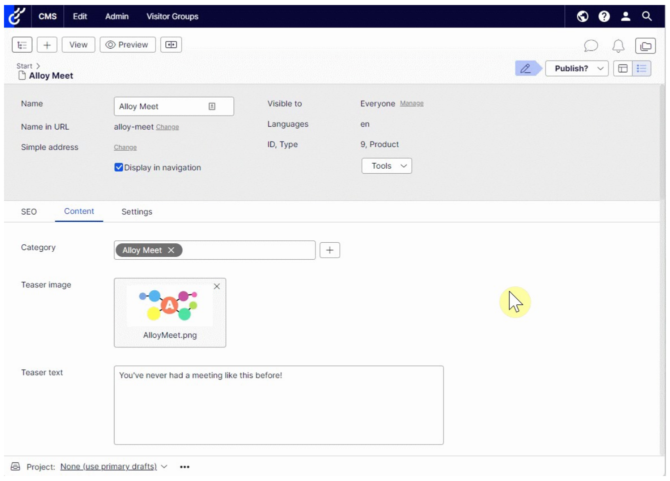


Sometimes you want to hide some properties in your CMS depending on the value of other properties. There are multiple ways to do this, but here is a rather simple approach that doesn't require advanced dojo skills.

Here is a very practical code snippet that you might find helpful (I can think of at least a handful situations in the past where I could have used this).

Sometimes you have properties in your edit-mode that might not be relevant some of the times the content type is used.

Maybe a content block has a boolean property that indicates whether it should show a link in the bottom or not? And if not, then there's no reason to confuse the editors with the link input.



You could also imagine scenarios where the block could display 1 or 2 columns - and if it's only going to display 1, then why bother with input for the other?!

In 2015 Greg made a great blog post showing how to accomplish all these sorts of customizations. But - being lazy, and a mere mortal - I was wondering if there was an even easier approach to doing this without even getting your hands dirty in dojo.

Turns out there is.

In the Code Gist below, I introduce a metadataextender that will run through all properties on your Content and if a property contains a new attribute called "[DependentProperty(propertyname, value)]" it will check for a property by that name - and only if the values match, will it show the dependent property in edit-mode.

This could fairly easily be extended to do many similar things.

Perhaps provide input to a selection factory, so the options in one dropdown lists depends on another drop-down list on the page?

Perhaps check against string values - or compare multiple dependent properties? The sky is the limit.

The trade-off here is that in order for this to work I also have to configure the property it's dependent on to reload the entire page on change - something that wouldn't be needed in the mystical world of dojo :-)

Also, I've kept it pretty simple - so we are not taking into account dependency-chains or similar complexities.

```
public class DependentPropertyAttribute : Attribute
{
   public string DependentProperty { get; set; }
   public object Value { get; set; }

   public bool Compare(object value)
   {
      if (value == null)
      {
         return Value == null;
      }

      return value.Equals(Value);
   }
```

```csharp

        public DependentPropertyAttribute(string propertyName, object value)
        {
            DependentProperty = propertyName;
            this.Value = value;
        }

    }
```

DependentPropertyAttribute.cs hosted with ❤ by GitHub　　view raw

```csharp
[ModuleDependency(typeof(InitializationModule))]
public class DependentPropertyInitialization : IInitializableModule
{
    public void Initialize(InitializationEngine context)
    {
        var registry = context.Locate.Advanced.GetInstance<MetadataHandlerRegistry>();
        registry.RegisterMetadataHandler(typeof(ContentData), new DependentPropertyMetadataExtender());
    }

    public void Uninitialize(InitializationEngine context) { }
}
```

DependentPropertyInitialization.cs hosted with ❤ by GitHub　　view raw

```csharp
public class DependentPropertyMetadataExtender : IMetadataExtender
{
    public void ModifyMetadata(ExtendedMetadata metadata, IEnumerable<Attribute> attributes)
    {
        if(metadata.Model is IContent)
        {
            //We have a piece of content. Let's iterate through the properties to see if there are any dependencies
            foreach (ExtendedMetadata extendedMetadata in metadata.Properties.OfType<ExtendedMetadata>())
            {
                var prop = extendedMetadata.Attributes.OfType<DependentPropertyAttribute>().FirstOrDefault();
                if(prop!=null)
                {
                    //We have a dependent property. Let's find the property that it depends on
                    var dp=metadata.Properties.OfType<ExtendedMetadata>().FirstOrDefault(em => em.PropertyName == prop.DependentProperty);
                    if (dp != null)
                    {
                        if (!prop.Compare(dp.InitialValue))
                        {
                            //Not same value, don't show!
                            extendedMetadata.ShowForEdit=false;
                        }
                        //Set reload on change
                        if (!dp.AdditionalValues.ContainsKey((object)"reloadOnChange"))
                        {
                            dp.AdditionalValues.Add((object)"reloadOnChange", (object)true);
                        }
                    }
                }
            }
        }
    }
}
```

DependentPropertyMetadataExtender.cs hosted with ❤ by GitHub　　view raw

```csharp
//Example of how to use the dependent properties in a content type.

//Here goes content type declaration typically
public class ProductPage : StandardPage, IHasRelatedContent
{

  //...Other properties on content type

   [Display(
   GroupName = SystemTabNames.Content,
   Order = 340)]
   public virtual bool ShowSpecialLink { get; set; }

   [Display(
   GroupName = SystemTabNames.Content,
   Order = 350)]
   [DependentProperty("ShowSpecialLink", true)]
   public virtual Url SpecialLink { get; set; }
}
```

ProductPage.cs hosted with ❤ by GitHub　　view raw

C# | Tips and Tricks | Optimizely (Episerver)

RECENT POSTS