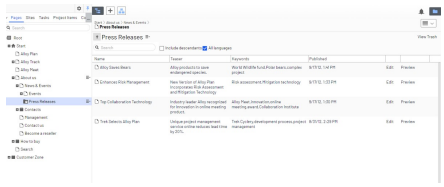
ALLAN THRAEN | 2 years ago | PDF |

Tips and Tricks .NET Development Optimizely (Episerver)

OPTIMIZELY GRIDVIEW CUSTOMIZATIONS



One of my favorite add-ons these days is the GridView. It's pretty customizable, but some of the customizations aren't the most well documented. Here's a couple of tricks I've found handy when using it.

For many years I've been fascinated by the general concept of content "buckets" - ways to maintain a site hierachy of content, but still acknowledge that some times you have a large amount of flat content items below a node - and you can improve the editorial experience significantly by treating it like a bucket full of content, rather than just a parent.

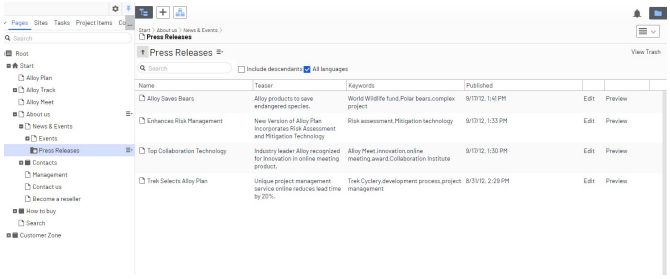
In fact, I wrote about it back in 2018 here.

For a while Episerver Labs (OptiLabs?) **have had** this great add-on **available** that introduces Grid View as an option through-out. I like it a lot and find myself using it more and more places. It's extremely versatile and powerful. And really - if you think about it, I'm sure you have places on your website where it would come in handy :-)

Here are a couple of typical customizations I've done:

Custom Properties in GridView Lists

The default view is quite good - but often when you show a list of custom content, there are probably a few properties you want to show in the gridview - and perhaps some of the default fields you don't want to show. Like in this example where I've setup a Gridview for the Alloy News page, and am showing the Teaser text and Keywords field directly on the view.



First off, to register the grid on the newspage, we have to do this:

```
using EPiServer.Labs.GridView;
using EPiServer.ServiceLocation;
using EPiServer.Shell;
using ExperimentSite.Models.Pages;

namespace ExperimentSite.Grid
{
    [ServiceConfiguration(typeof(ViewConfiguration))]
    public class ProductDescriptionGroupPageGridView : SearchContentView<NewsPage>
    {
    }
}
```

Then, we have to make sure that the fields we want to show on the grid have the proper attribute in the model:

```
        Display(
                GroupName = Global.GroupNames.MetaData,
        [       Order = 200)                        ]
        [CultureSpecific]
        [BackingType(typeof(PropertyStringList))]
        [IncludeInQueryResult] //Needed to show column in grid view
        public virtual string[] MetaKeywords { get; set; }
```

And finally we need to setup a UIDescriptor where we will build the list of fields to show. Notice the 2 different ways of selecting a custom property (one gives slightly more control than the other):

```
using EPiServer.Labs.GridView;
using EPiServer.Labs.GridView.GridConfiguration;
using EPiServer.Shell;
using ExperimentSite.Models.Pages;

namespace ExperimentSite.Grid
{
    [UIDescriptorRegistration]
    public class NewsPageUIDescriptor : ExtendedUIDescriptor<NewsPage>
    {
        public NewsPageUIDescriptor()
                    : base()
                                {
            //Setting it as the default view
            DefaultView = SearchContentViewContentData.ViewKey;

            //Specifying columns
```

```
            GridSettings = new GridSettings
            {
                Columns = new ColumnsListBuilder()
                .WithContentName()
                .WithColumn("teaser", displayName: "Teaser", formaterType: "property", propertyName: "
                .WithRawPropertyValue("Keywords","MetaKeywords")
                //.WithContentStatus()
                //.WithContentTypeName()
                //.WithVisibleInMenu()
                //.WithCreatedBy()
                .WithPublishDate()
                .WithEdit()
                .WithPreviewUrl()
                .WithActionMenu()
                .Build()
            };
        }
    }
}
```

Custom Root when using the GridView Property Selector

Another really useful customization is when you need to make the GridView available in a property selector - for example to pick elements for a content area (although it works in pretty much any property that holds references to content). You typically want to set the proper starting points for it.

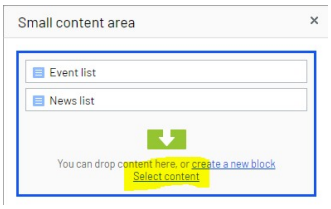
For example - a content area where an editor can show news listings, the gridview selector should start at the grid root.

This can be done by adding a couple of attributes to the property definition - as here on the Alloy Product Page:

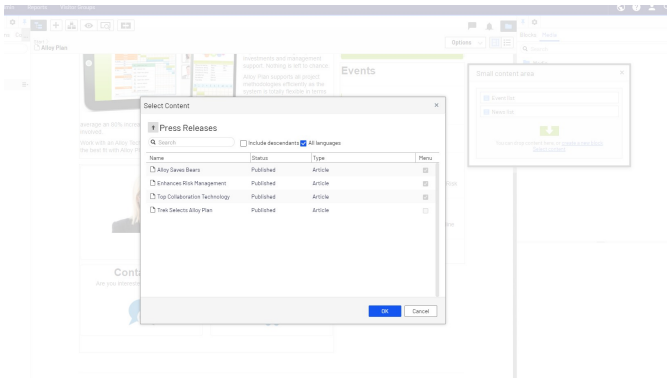
```
    public class ProductPage : StandardPage, IHasRelatedContent
    {
        [Required]
        [Display(Order = 305)]
        [UIHint(Global.SiteUIHints.StringsCollection)]
        [CultureSpecific]
        public virtual IList<string> UniqueSellingPoints { get; set; }

         Display(
                    GroupName = SystemTabNames.Content,
        [           Order = 330)                      ]
        [CultureSpecific]
        [AllowedTypes(new[] { typeof(IContentData) },new[] { typeof(JumbotronBlock) })]
        //Below lines are needed to allow Related Content Area to use the grid view selector
        [UIHint(GridViewEditing.UIHint)]
        [PropertyGridConfiguration]
        public virtual ContentArea RelatedContentArea { get; set; }
    }
```

Which add's the link to Select Content for editors.



When they select content, they'll get a gridview and this is what we want to control.



It's really pretty simple - just register a GridviewRootsResolver.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using EPiServer;
using EPiServer.Core;
using EPiServer.Labs.GridView.EditorDescriptors;
using EPiServer.ServiceLocation;
using ExperimentSite.Models.Pages;

namespace ExperimentSite.Widgets
{
    [ServiceConfiguration(typeof(IChildrenGridPropertyRootResolver))]
    public class GridviewRootsResolver : IChildrenGridPropertyRootResolver
    {
```

```
        public GridviewRootsResolver()
                                    {
        }

        public ContentReference GetRoot(IContent content, string propertyName)
                                                                        {
            //If this is being called from the "related" content area on a product page
            if (propertyName.Equals(nameof(ProductPage.RelatedContentArea)) && content is ProductPage
            {
                return new ContentReference(16); //Hardcoded for now - but could be looked up using a
            }
            //Handle other cases where the gridview root should be resolved

            return null;
        }
    }
}
```

This should be it - but I struggled a little bit getting it to work, and it turned out that the dependency injection prefered the default rootsresolver to mine - until I forced it like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using EPiServer.Framework;
using EPiServer.Framework.Initialization;
using EPiServer.Labs.GridView.EditorDescriptors;
using EPiServer.ServiceLocation;

namespace ExperimentSite.Widgets
{
    [InitializableModule]
    [ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
    public class GridViewInit : IConfigurableModule
    {
        public void ConfigureContainer(ServiceConfigurationContext context)
                                                                    {
            context.Services.AddTransient<IChildrenGridPropertyRootResolver, GridviewRootsResolver>();
        }

        public void Initialize(InitializationEngine context)
                                                        {
        }

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

Tips and Tricks | .NET Development | Optimizely (Episerver)

RECENT POSTS

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