Embrace, don't Replace
The Umbraco Way

Lennard Fonteijn

4NG

We grow digital business.

verb: **embrace** /ɪmˈbreɪs,ɛmˈbreɪs/
1. an act of accepting something willingly or enthusiastically.

Embrace

2. it's only two letters away from Umbraco.

Lennard Fonteijn

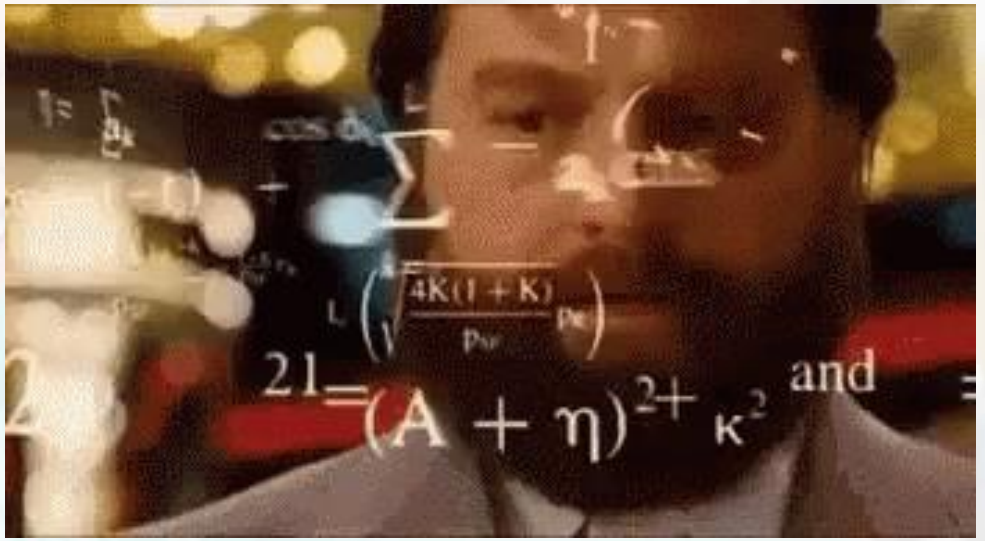
CTO @ Arlanet part of 4NG

Good to know:

- Always playing with technology, over 20 years of (professional) experience.
- Employed at Arlanet part of 4NG for about 10 years.
- Employed at the Amsterdam University of Applied Sciences as a Software Engineering docent for about 7 years.
- Umbraco MVP & Umbraco Certified Master
- Nocturnal animal

What to expect from this talk?

- A little context about what sparked this talk
- Code, lots of code...
- Takeaways to apply in your own projects



The Hangover

The case

- Custom “CMS” to Umbraco
- Developers with very little experience
- Consultancy only

Not the actual client

The challenge

For for the project

- One instance, many clients.
- Strict data segregation between clients.
- 25 languages globally, but clients only use a subset.
- Lots of configurable bits by administrators, but not clients.

The challenge

For for the developers

Achieve the **most**, with the **least** amount of code possible.

Genuine developer reaction

Establishing the rules

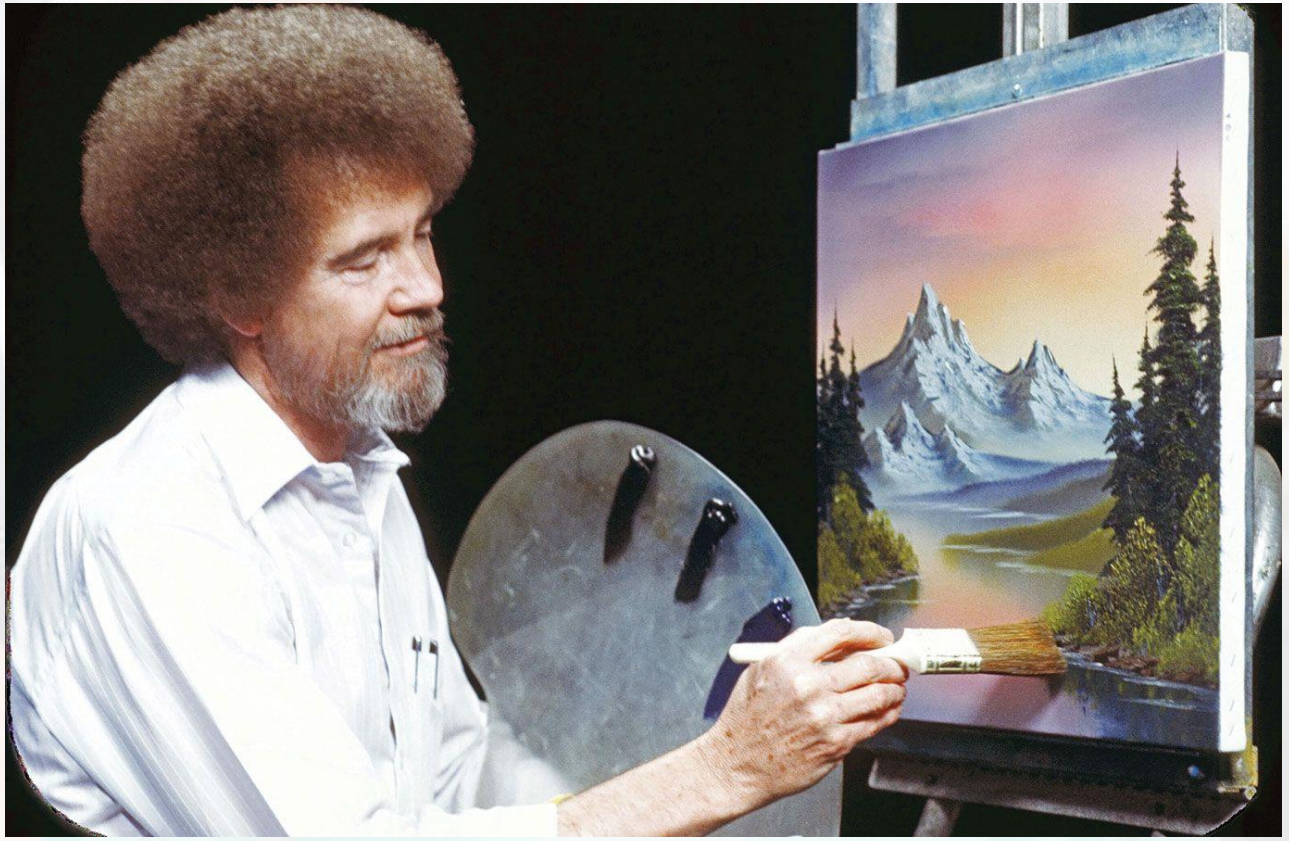
Keeping things workable for all experience levels

Rule #1

Give the out-of-the-box Umbraco experience a chance.

Rule #2

Only use extension points provided by Umbraco.



"A happy little AngularJS hack here,
some Reflection there."
– Bob Ross

Rule #3

KISS!

- Keep
- It
- Simple
- SERIOUSLY

Applying the rules

Everyone can make rules, actually following them is a second.

The rules applied

A lot of problems can be solved using **Block List**

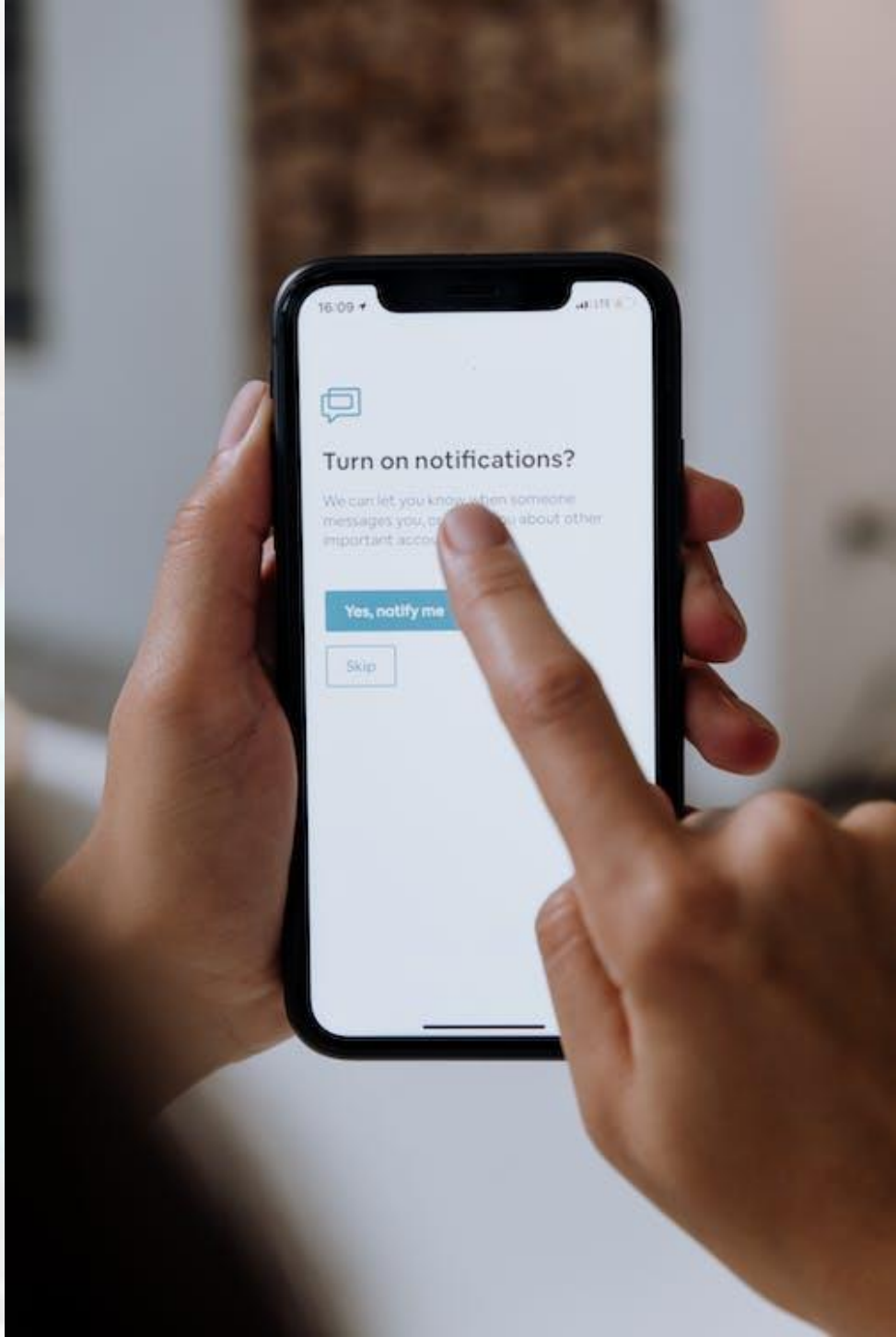
Otherwise, **Contentment** probably can.

Thanks Lee

The rules applied

Use **Notification Handlers** whenever possible

Only override **services** if you really have to.



The rules applied

Always keep the **end-user** in mind

The rules applied

Always keep the **CMS Updates** in mind

The rules applied

Always keep the **developer** in mind

Overarching goal for end-users

Someone with zero web-editing experience should feel safe to press buttons*

* And not mess things up in the process...

What does that mean for the end-user?

Hide UI elements that don’t concern them

What does that mean for the end-user?

Everything should have clear and concise namings and descriptions

What does that mean for the end-user?

Everything should be translated/translatable

Overarching goal for developers

The learning curve of the code-base can’t be too steep

What does that mean for the developer?

Beginner

Notification Handlers

- SendingContentNotification
- TreeNodesRenderingNotification
- SendingAllowedChildrenNotification
- ContentSavingNotification (et al.)
- ServerVariablesParsingNotification

Composers / Components

- Or UmbracoApplicationStartingNotification (et al.) instead of Component

What does that mean for the developer?

Intermediate

- Extending **Examine**
- Adding **Contentment** datasources
- Custom Url Provider (**IUrlProvider**)
- Tree Actions (**IAction**)
- Custom File Systems (eg. **IMediaPathScheme**)
- Filters / Middleware

What does that mean for the developer?

Advanced

Overriding (internal) services containing non-virtual methods using proxies and interfaces

Beginner

Finally, some code!

Notification Handlers 101

The bread and butter of extending Umbraco

```
1  public class ExampleNotificationHandler :
2      INotificationHandler<ContentSavingNotification>,
3      INotificationAsyncHandler<ContentSavingNotification>
4  {
5      private readonly ILogger<ExampleNotificationHandler> _logger;
6
7      public ExampleContentNotificationHandler(
8          ILogger<ExampleContentNotificationHandler> logger
9      )
10     {
11         _logger = logger;
12     }
13
14     //...
15 }
```

Notification Handlers 101

The bread and butter of extending Umbraco

```
1  public class ExampleNotificationHandler :
2      INotificationHandler<ContentSavingNotification>,
3      INotificationAsyncHandler<ContentSavingNotification>
4  {
5      //...
6
7      public void Handle(ContentSavingNotification notification)
8      {
9          foreach (var content in notification.SavedEntities)
10         {
11             _logger.LogInformation("Saving content {Name}", content.Name);
12         }
13     }
14 }
```

Notification Handlers 101

The bread and butter of extending Umbraco

```
1 public class ExampleNotificationHandler :
2     INotificationHandler<ContentSavingNotification>,
3     INotificationAsyncHandler<ContentSavingNotification>
4 {
5     //...
6
7     public Task HandleAsync(
8         ContentSavingNotification notification,
9         CancellationToken cancellationToken
10    )
11    {
12        foreach (var content in notification.SavedEntities)
13        {
14            _logger.LogInformation("Saving content {Name}", content.Name);
15        }
16
17        return Task.CompletedTask;
18    }
19  }
```

Notification Handlers 101

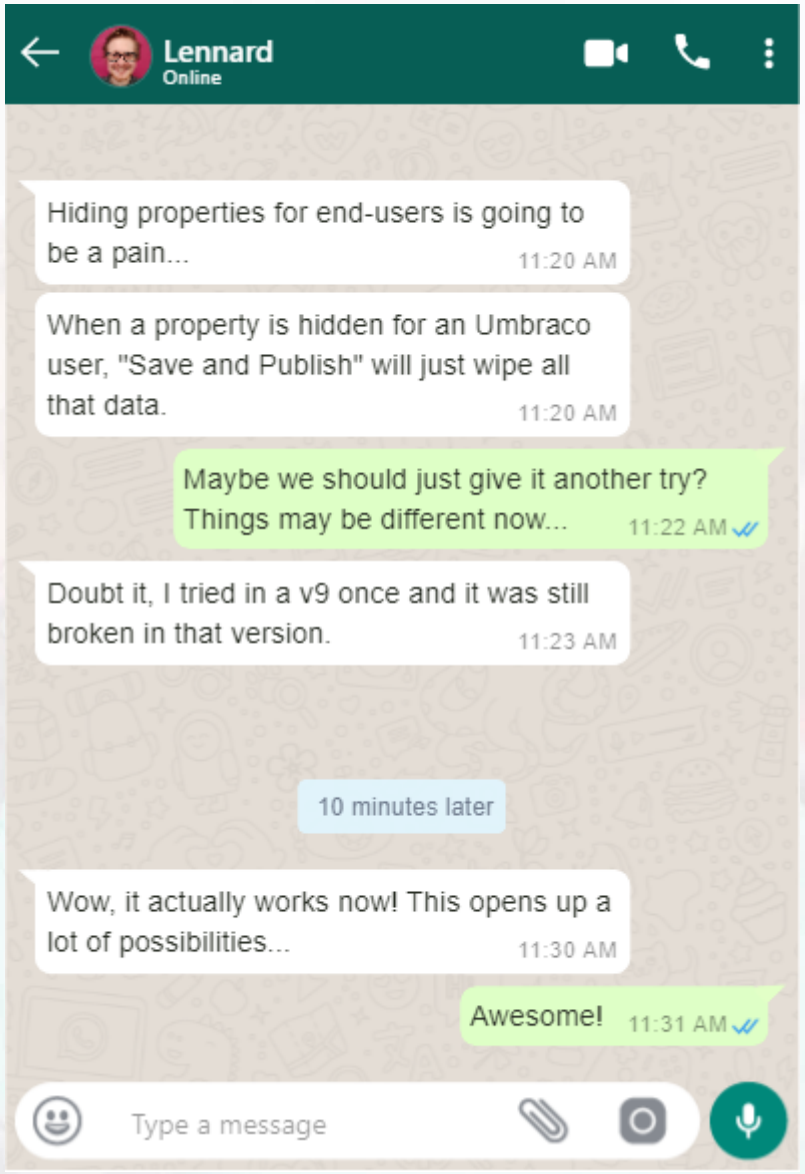
The bread and butter of extending Umbraco

```
1  public class StartupComposer : IComposer
2  {
3      public void Compose(IUmbracoBuilder builder)
4      {
5          builder.AddNotificationHandler<
6              ContentSavingNotification,
7              ExampleNotificationHandler
8          >();
9          builder.AddNotificationAsyncHandler<
10             ContentSavingNotification,
11             ExampleNotificationHandler
12         >();
13     }
14 }
```

Hiding Properties

"Hiding properties is broken and wipes data!"

Says ~~Lennard~~ **a developer** based on previous experiences.



Digital re-enactment of true events

Hiding Properties

It’s not broken anymore!

```
1 public class HidePropertyNotificationHandler : INotificationHandler<SendingContentNotification>
2 {
3     private static readonly HashSet<string> _hideProperties = new()
4     {
5         "hideTest"
6     };
7
8     //...
9 }
```

Hiding Properties

It’s not broken anymore!

```
public class HidePropertyNotificationHandler : INotificationHandler<SendingContentNotification>
{
    //...

    public void Handle(SendingContentNotification notification)
    {
        if (notification.Content.DocumentType?.Alias != Home.ModelTypeAlias)
        {
            return;
        }

        notification.Content.Variants = notification.Content.Variants.Select(x =>
        {
            x.Tabs = x.Tabs.Select(y =>
            {
                y.Properties = y.Properties?
                    .Where(z => !_hideProperties.Contains(z.Alias));

                return y;
            });

            x.Tabs = x.Tabs
                .Where(t => t.Properties != null && t.Properties.Any());

            return x;
        });
    }
}
```

Hiding Properties

It’s not broken anymore!

```
1 public class StartupComposer : IComposer
2 {
3     public void Compose(IUmbracoBuilder builder)
4     {
5         builder.AddNotificationHandler<
6             SendingContentNotification, HidePropertyNotificationHandler
7         >();
8     }
9 }
```

Hiding Properties

It’s not broken anymore!

```
1 public abstract class NodeDisplaySetting
2 {
3     public class RoleSetting
4     {
5         public bool ExcludeProperties { get; init; }
6         public IEnumerable<string> Properties { get; init; }
7     }
8
9     public abstract string ContentTypeAlias { get; set; }
10
11    public abstract Dictionary<EEditorRole, RoleSetting> RoleSettings { get; }
12 }
```

Adjusting the language dropdown

Even more Notification Handler goodness

```
1  public class AdjustLanguageDropdownNotificationHandler :
2      INotificationHandler<SendingContentNotification>
3  {
4      private readonly IContentTypeService _contentTypeService;
5      private readonly IBackOfficeSecurity _backOfficeSecurity;
6
7      private readonly Dictionary<string, string> _userGroupToLanguage = new()
8      {
9          { "editorDutch", "nl-NL" },
10         { "editorEnglish", "en-US" },
11         { "editorGerman", "de-DE" }
12     };
13
14     public AdjustLanguageDropdownNotificationHandler(
15         IContentTypeService contentTypeService,
16         IBackOfficeSecurity backOfficeSecurity
17     )
18     {
19         _contentTypeService = contentTypeService;
20         _backOfficeSecurity = backOfficeSecurity;
21     }
22
23     //...
24 }
```

Adjusting the language dropdown

Even more Notification Handler goodness

```
{
    //...

    public void Handle(SendingContentNotification notification)
    {
        var contentType = _contentTypeService.Get(notification.Content.ContentTypeKey);

        if (
            contentType == null ||
            !contentType.VariesByCulture() ||
            !notification.Content.ParentId.HasValue
        )
        {
            return;
        }

        // ...
    }
}
```

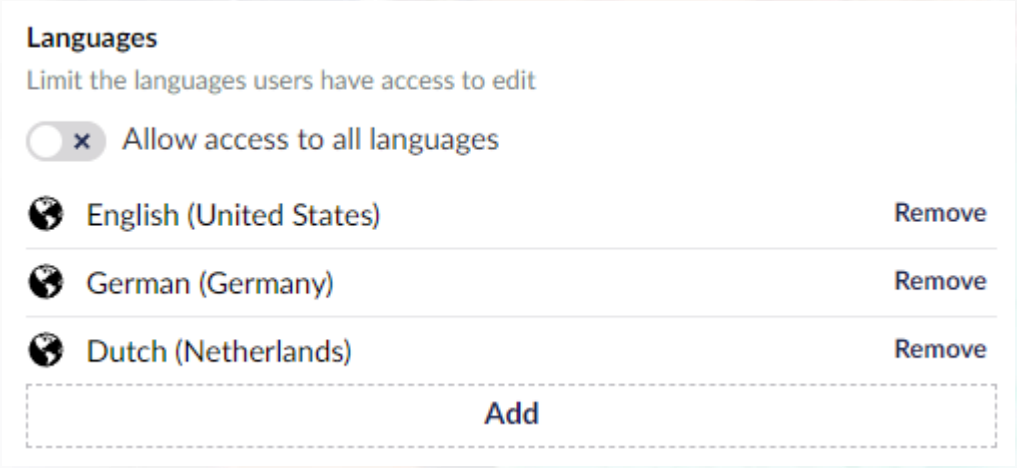
Adjusting the language dropdown

Even more Notification Handler goodness

```
{
    //...

    public void Handle(SendingContentNotification notification)
    {
        //...

        var userGroups = _backOfficeSecurity.CurrentUser.Groups
            .Select(x => x.Alias)
            .ToHashSet();

        var cultures = _userGroupToLanguage
            .Where(x => userGroups.Contains(x.Key))
            .Select(x => x.Value)
            .ToHashSet();

        notification.Content.Variants =
            notification.Content.Variants
                .Where(x =>
                    x.Language != null &&
                    cultures.Contains(x.Language.IsoCode)
                );
    }
}
```

Adjusting the language dropdown

And then you find out, Umbraco v12 has native support for this now... (ノ°□°)ノ ⌒ ┻━┻



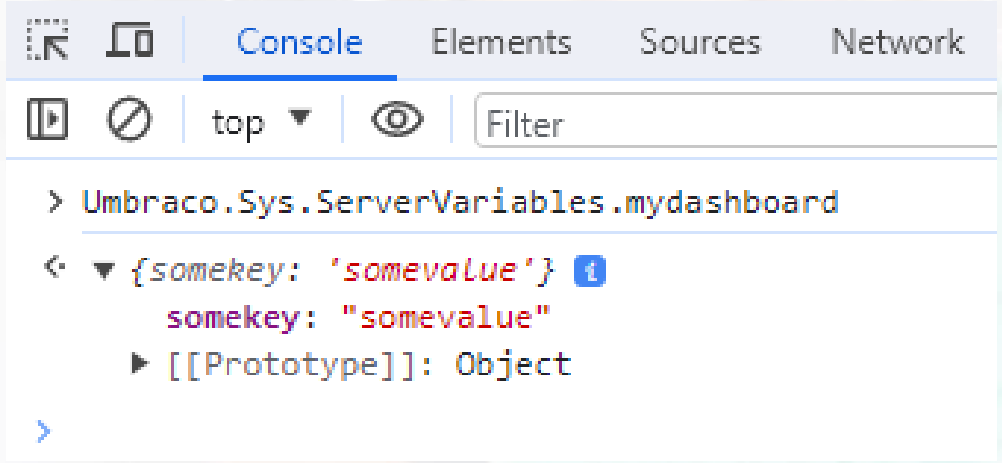
Server Variables

Useful for passing information from the backend to the Backoffice

```
1  public class ServerVariablesNotificationHandler :
2      INotificationHandler<ServerVariablesParsingNotification>
3  {
4      public void Handle(ServerVariablesParsingNotification notification)
5      {
6          notification.ServerVariables["mydashboard"] = new Dictionary<string, string>
7          {
8              { "somekey", "somevalue" }
9          };
10     }
11 }
```

Server Variables

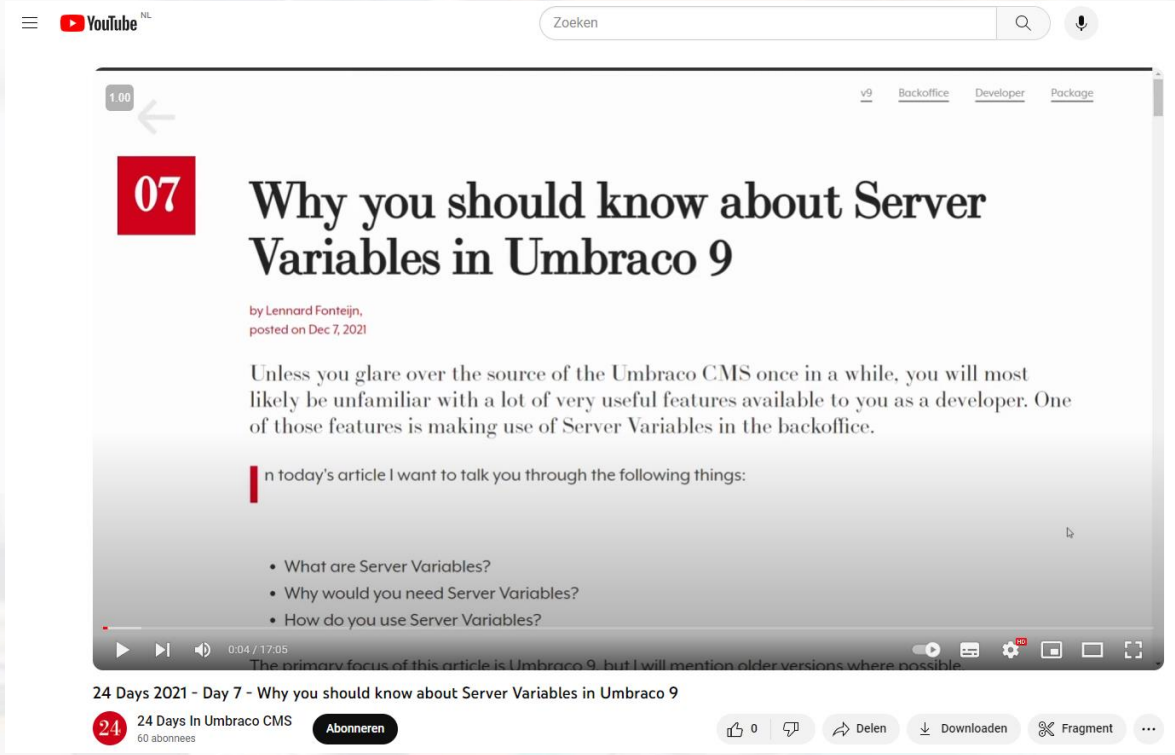
Useful for passing information from the backend to the Backoffice



Server Variables

I actually wrote an extensive article about this for 24days.in

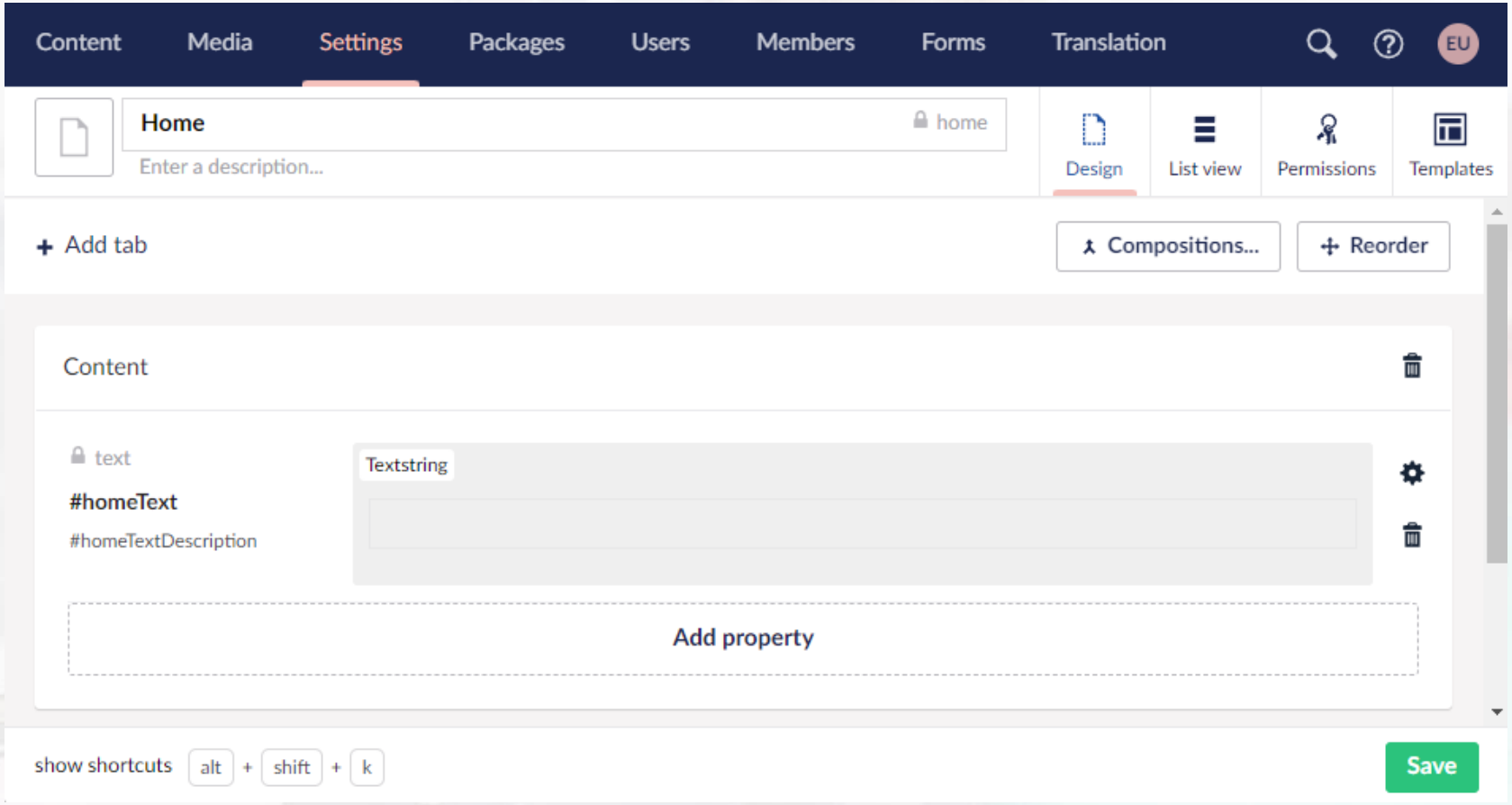
https://archive.24days.in/umbraco-cms/2021/umbraco-9-server-variables/



Reading not your thing? Let **Paul Seal** read it to you instead!
https://www.youtube.com/watch?v=VPIndLWz0RM

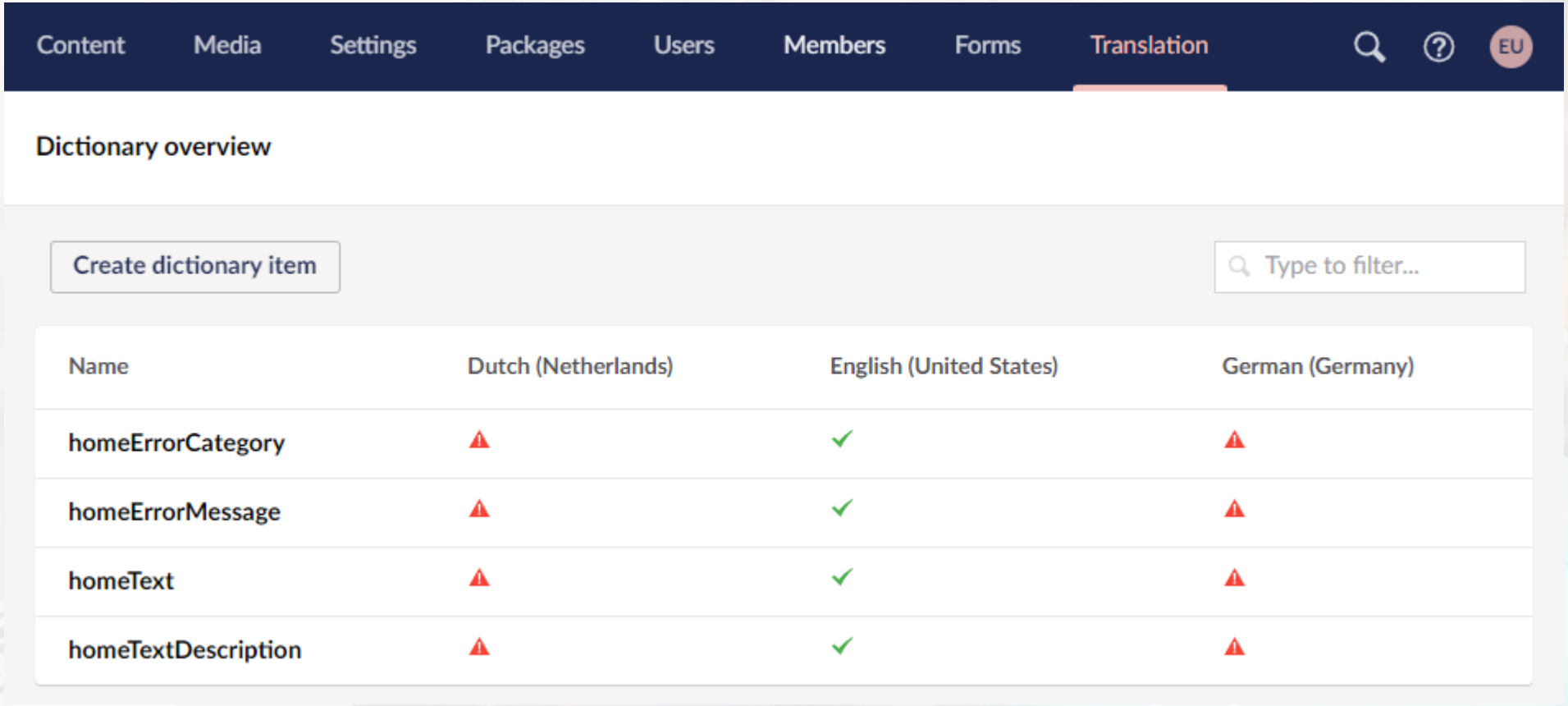
Translated Properties

Did you know you could do this?



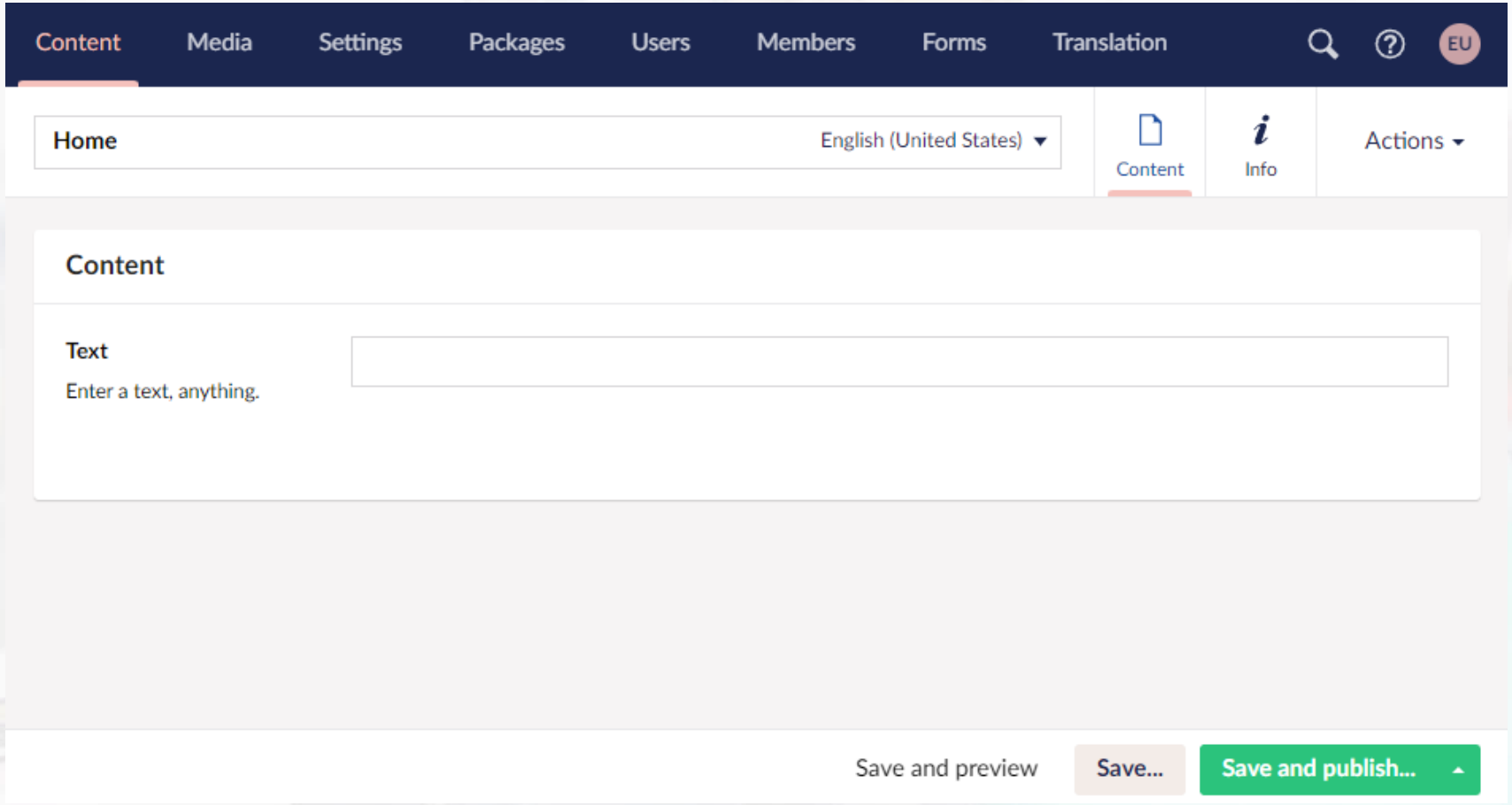
Translated Properties

Did you know you could do this?



Translated Properties

Did you know you could do this?



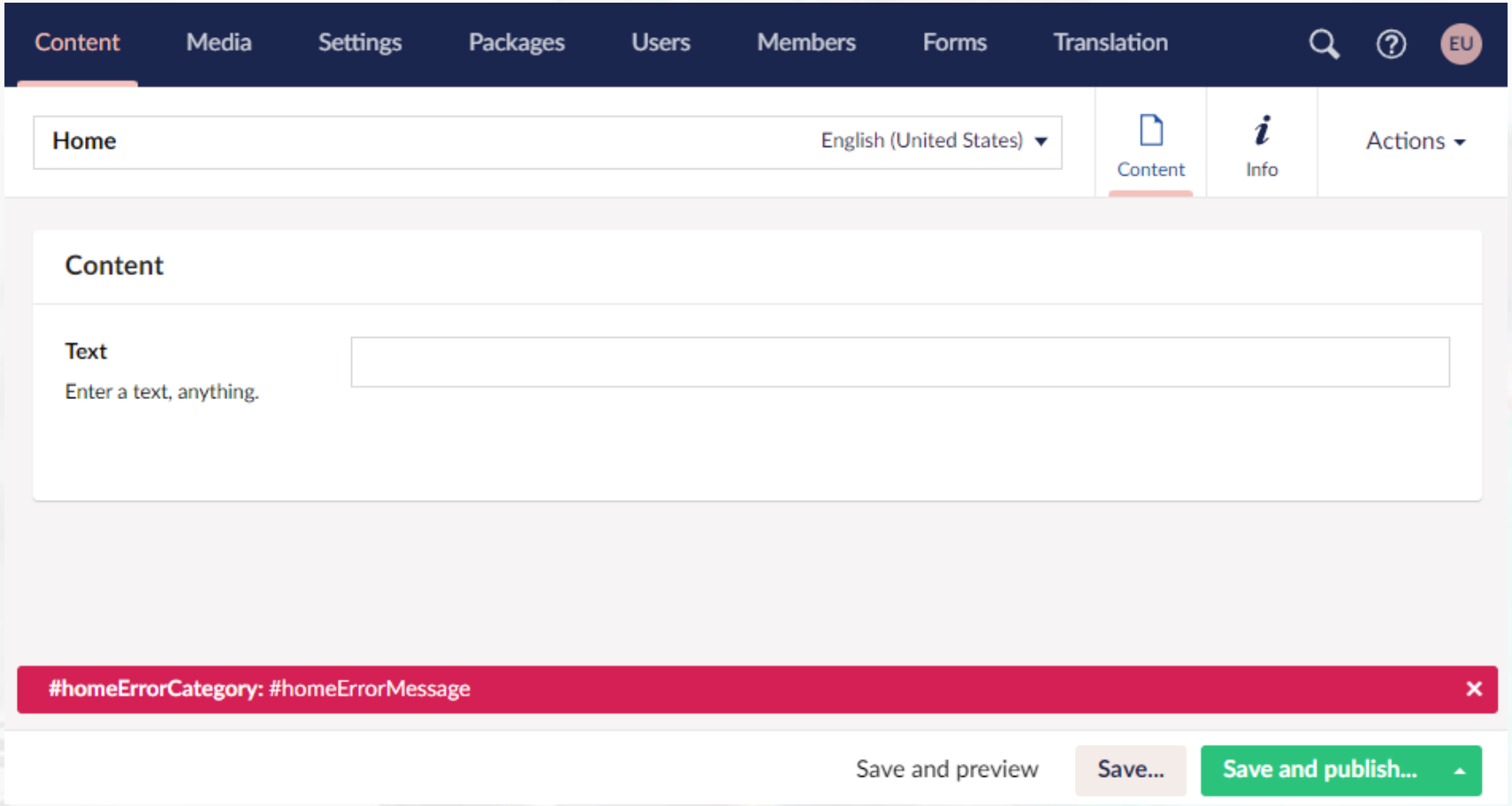
Translated Properties

But does it also work for Notification Handler messages?

```
public class PublishHomeNotificationHandler :
    INotificationHandler<ContentPublishingNotification>
{
    public void Handle(ContentPublishingNotification notification)
    {
        foreach (var content in notification.PublishedEntities)
        {
            if (content.ContentType.Alias != Home.ModelTypeAlias)
            {
                continue;
            }

            var value = content.Properties["text"]?.GetValue() as string;

            if (!string.IsNullOrWhiteSpace(value))
            {
                continue;
            }

            notification.Cancel = true;
            notification.Messages.Add(new EventMessage(
                    "#homeErrorCategory",
                    "#homeErrorMessage",
                    EventMessageType.Error
                )
                {
                    IsDefaultEventMessage = true
                }
            );
        }
    }
}
```

Translated Properties

It does not 😱



Intermediate

Things are heating up!

Adding and replacing services

Dependency Injection you say?

```
1 public interface ITestService
2 {
3     string Test();
4 }
5
6 public class TestService : ITestService
7 {
8     public string Test()
9     {
10        return "This is a test!";
11    }
12 }
```

Adding and replacing services

Dependency Injection you say?

```
1  public class StartupComposer : IComposer
2  {
3      public void Compose(IUmbracoBuilder builder)
4      {
5          builder.Services.AddTransient<ITestService, TestService>();
6          //OR
7          builder.Services.AddScoped<ITestService, TestService>();
8          //OR
9          builder.Services.AddSingleton<ITestService, TestService>();
10     }
11 }
```

Adding and replacing services

Dependency Injection you say?

```
1  public class TestApiController : UmbracoApiController
2  {
3      private readonly ITestService _testService;
4
5      public TestApiController(ITestService testService)
6      {
7          _testService = testService;
8      }
9
10     [HttpGet]
11     public IActionResult Test()
12     {
13         return Content(_testService.Test());
14     }
15 }
```

Extending Examine

Really useful if you don’t want to go the custom route

```
1  public class ExamineComponent : IComponent
2  {
3      private readonly IExamineManager _examineManager;
4
5      public ExamineComponent(IExamineManager examineManager)
6      {
7          _examineManager = examineManager;
8      }
9
10     //...
11 }
```

Extending Examine

Really useful if you don't want to go the custom route

```
public class ExamineComponent : IComponent
{
    //...

    public void Initialize()
    {
        if (
            _examineManager.TryGetIndex(
                Constants.UmbracoIndexes.InternalIndexName,
                out var internalIndex
            ) &&
            internalIndex is LuceneIndex internalUmbracoContentIndex
        )
        {
            internalUmbracoContentIndex.TransformingIndexValues +=
IndexProviderTransformingIndexValues;
        }
    }

    //...
}
```

Extending Examine

Really useful if you don't want to go the custom route

```
1  public class ExamineComponent : IComponent
2  {
3      //...
4
5      private void IndexProviderTransformingIndexValues(object sender, IndexingItemEventArgs e)
6      {
7          if (e.ValueSet.Category != IndexTypes.Content)
8          {
9              return;
10         }
11
12         var values = e
13             .ValueSet
14             .Values
15             .ToDictionary(
16                 x => x.Key,
17                 x => (IEnumerable<object>)x.Value.ToList()
18             );
19
20         values["somekey"] = new List<object> { "somevalue" };
21
22         e.SetValues(values);
23     }
24 }
```

Extending Examine

Really useful if you don't want to go the custom route

```
1  public class ExamineIndexOptions : IConfigureNamedOptions<LuceneDirectoryIndexOptions>
2  {
3      public void Configure(LuceneDirectoryIndexOptions options)
4      {
5          Configure(string.Empty, options);
6      }
7
8      public void Configure(string name, LuceneDirectoryIndexOptions options)
9      {
10         switch (name)
11         {
12             case Constants.UmbracoIndexes.InternalIndexName:
13                 options.FieldDefinitions.TryAdd(
14                     new FieldDefinition("somekey", FieldDefinitionTypes.Raw)
15                 );
16                 break;
17         }
18     }
19 }
```

Extending Examine

Really useful if you don't want to go the custom route

```
1 public class StartupComposer : IComposer
2 {
3     public void Compose(IUmbracoBuilder builder)
4     {
5         builder.Components().Append<ExamineComponent>();
6         builder.Services.ConfigureOptions<ExamineIndexOptions>();
7     }
8 }
```

Contentment datasources

Even more useful than Block List

```
1 public class TestDataSource : IDataSourceValueConverter, IDataListSource
2 {
3     public string Name => "Test DataSource";
4     public string Description => "Test DataSource description";
5     public string Icon => "icon-umb-media";
6     public string Group => "Custom";
7
8     public OverlaySize OverlaySize => OverlaySize.Small;
9
10    public Dictionary<string, object> DefaultValues => new();
11    public IEnumerable<ConfigurationField> Fields => Array.Empty<ConfigurationField>();
12
13    public Type GetValueType(Dictionary<string, object> config)
14    {
15        return typeof(string);
16    }
17
18    public object ConvertValue(Type type, string value)
19    {
20        return value;
21    }
22
23    //...
24 }
```

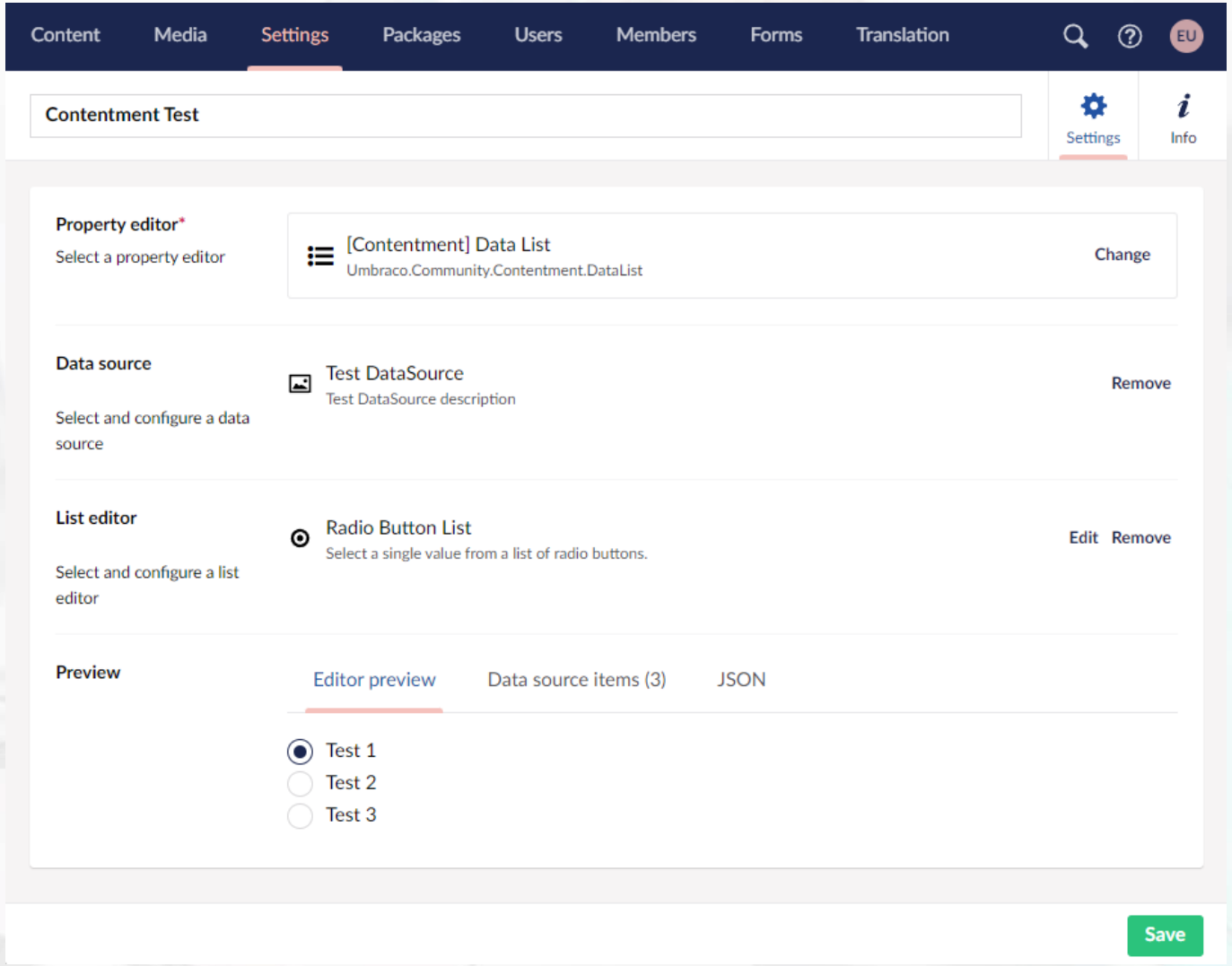
Contentment datasources

Even more useful than Block List

```
public class TestDataSource : IDataSourceValueConverter, IDataListSource
{
    //...

    public IEnumerable<DataListItem> GetItems(Dictionary<string, object> config)
    {
        return new[]
        {
            new DataListItem
            {
                Name = "Test 1",
                Value = "test-1"
            },
            new DataListItem
            {
                Name = "Test 2",
                Value = "test-2"
            },
            new DataListItem
            {
                Name = "Test 3",
                Value = "test-3"
            }
        };
    }
}
```

Contentment datasources

Even more useful than Block List



Filters and Middleware

Modifying Startup.cs without actually modifying Startup.cs

```
1  public class TestMiddleware
2  {
3      private readonly RequestDelegate _next;
4
5      public TestMiddleware(RequestDelegate next)
6      {
7          _next = next;
8      }
9
10     public async Task Invoke(HttpContext context)
11     {
12         if (context.Request.Path.StartsWithSegments("/ping"))
13         {
14             context.Response.StatusCode = 200;
15             context.Response.WriteAsync("Pong!");
16
17             return;
18         }
19
20         await _next(context);
21     }
22 }
```

Filters and Middleware

Modifying Startup.cs without actually modifying Startup.cs

```
1  public class TestPipelineFilter : UmbracoPipelineFilter
2  {
3      public TestPipelineFilter()
4          : base(nameof(TestPipelineFilter))
5      {
6          PrePipeline = builder =>
7          {
8              builder.UseMiddleware<TestMiddleware>();
9          };
10     }
11 }
```

Filters and Middleware

Modifying Startup.cs without actually modifying Startup.cs

```
1  public class StartupComposer : IComposer
2  {
3      public void Compose(IUmbracoBuilder builder)
4      {
5          builder.Services.Configure<UmbracoPipelineOptions>(options =>
6          {
7              options.AddFilter(new TestPipelineFilter());
8          });
9      }
10 }
```

Advanced

It’s like magic!

4NG

Replacing internal services

Not because we have to, but because we can.

```
namespace Umbraco.Cms.Core.Services;

internal class UserService : RepositoryService, IUserService
{
    //...

    public IUser? GetByEmail(string email)
    {
        using (ICoreScope scope = ScopeProvider.CreateCoreScope(autoComplete: true))
        {
            IQuery<IUser> query = Query<IUser>().Where(x => x.Email.Equals(email));
            return _userRepository.Get(query)?.FirstOrDefault();
        }
    }

    //...
}
```

Replacing internal services

Not because we have to, but because we can.

```
1 public class MyUserService : UserService
2 {
3     //Is not possible :(
4 }
```

Replacing internal services

Not because we have to, but because we can.

```
1 public class MyUserService : IUserService
2 {
3     //...
4 }
```

Replacing internal services

Not because we have to, but because we can.

```
1 public class ProxyUserService : IUserService
2 {
3     private readonly IUserService _originalUserService;
4
5     public ProxyUserService(IUserService originalUserService)
6     {
7         _originalUserService = originalUserService;
8     }
9
10    public int GetCount(MemberCountType countType)
11    {
12        return _originalUserService.GetCount(countType);
13    }
14
15    //...
16 }
```

Replacing internal services

Not because we have to, but because we can.

```
1 public class StartupComposer : IComposer
2 {
3     public void Compose(IUmbracoBuilder builder)
4     {
5         var originalUserService = builder.Services
6             .First(x => x.ServiceType == typeof(IUserService))
7             .ImplementationType!;
8
9         builder.Services.AddUnique<IUserService>(provider =>
10            new ProxyUserService(
11                (IUserService)provider.CreateInstance(originalUserService)
12            )
13        );
14    }
15 }
```

Replacing internal services

Not because we have to, but because we can.

```
1 public class MyUserService : ProxyUserService
2 {
3     public MyUserService(IUserService originalUserService)
4         : base(originalUserService)
5     {
6         //Do nothing
7     }
8
9     public new IUser GetByEmail(string email)
10    {
11        throw new NotImplementedException();
12    }
13 }
```

Replacing internal services

Not because we have to, but because we can.

```
1  public class MyUserService : ProxyUserService, IMembershipMemberService<IUser>
2  {
3      public MyUserService(IUserService originalUserService)
4          : base(originalUserService)
5      {
6          //Do nothing
7      }
8
9      IUser IMembershipMemberService<IUser>.GetByEmail(string email)
10     {
11         return base.GetByEmail(email);
12     }
13 }
```

Replacing internal services

Not because we have to, but because we can.

```
1  public class StartupComposer : IComposer
2  {
3      public void Compose(IUmbracoBuilder builder)
4      {
5          //...
6
7          builder.Services.AddUnique<IUserService>(provider =>
8              new MyUserService(
9                  (IUserService)provider.CreateInstance(originalUserService)
10             )
11         );
12     }
13 }
```

Closing thoughts

What did we learn?

Rules like these keep you on your toes.

Closing thoughts

What is your takeaway from this talk?

Let me know! (Pretty please?)

4NG

Feedback or questions? Just reach out!

lennard@arlanet.com

LennardF1989

https://github.com/LennardF1989/EmbraceUmbraco