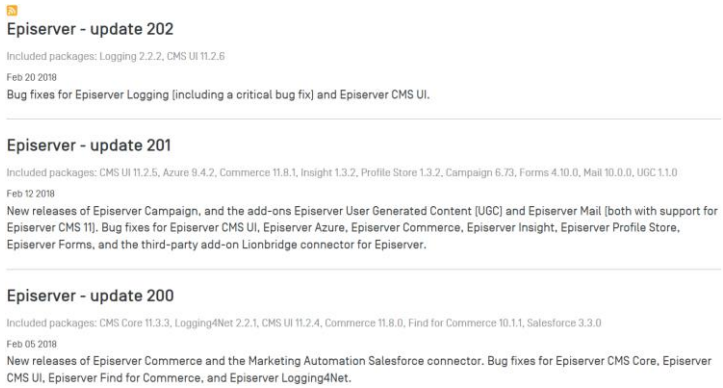
Course title: *Episerver CMS – Advanced Development* Course code: 170-3030

Course version: 18.02, 25th February 2018 Product Update 202, 20th February 2018

Episerver CMS Visual Studio Extension version: 11.1.0.326

Episerver CMS packages: EPiServer.CMS.Core 11.3.3, EPiServer.CMS.UI 11.2.6

http://world.episerver.com/releases/



Episerver CMS Advanced Development

Introduction

In this course we go deeper into the concepts that were introduced in the fundamentals course and explore more specialized parts of the Episerver platform.

Prerequisites are attendance on *Episerver CMS Development Fundamentals* or equivalent experience.

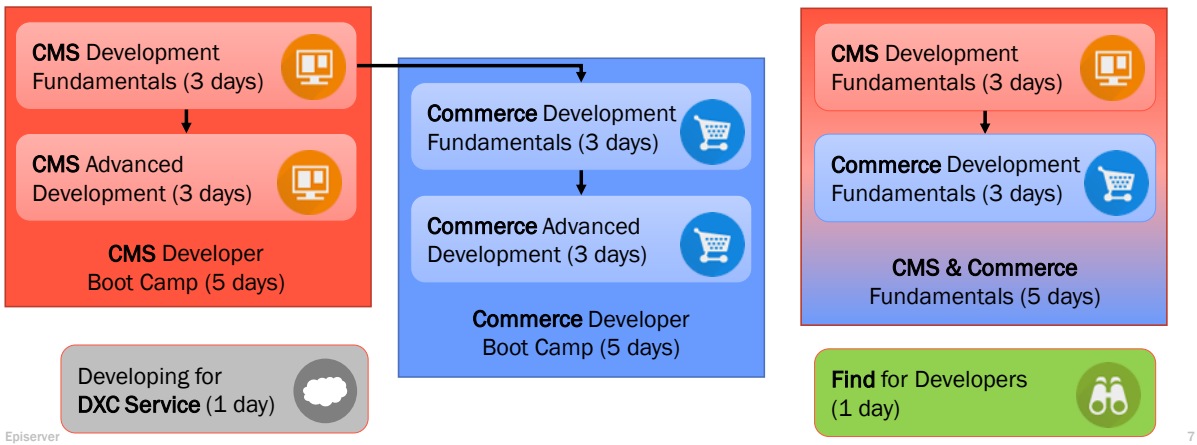
Episerver

Introduction – About the course

Education Services offers developer course tracks. To attend this course you should have skills equivalent to *CMS Development Fundamentals*.

Course tracks

Developer courses
http://www.episerver.com/services/education/services-education-courses/developers/



Episerver 7

Introduction – About the course

Course agenda

Timings for each module will vary based on:
- If the course is delivered as a 3-day course, or as the second part of *CMS Developer Boot Camp*.
- The experience and interest of participants.
- How many of the exercises participants complete.

- **Introduction** ½ hour
- Module A: **Reviewing Episerver CMS Fundamentals** 1½ - 3 hours
- Module B: **Working with Content using APIs** 1½ - 3 hours
- Module C: **Integrating Data and Forms** 1 - 4 hours
- Module D: **Customizing Properties for Editors** 1 - 4 hours
- Module E: **Rendering, Personalizing, and Indexing Content** 1 - 4 hours
- Module F: **Extending with Plug-ins and Add-ons** 1 - 4 hours

- Optional modules
  - Module G: **Episerver Find** 1 - 3 hours
  - Module H: **Episerver Social** 1 - 3 hours

Episerver 8

Introduction – About the course

About the course exercises

You do not have to complete every exercise during the course! You can do some after the course.

Unlike the *CMS Development Fundamentals* course that builds up a complete site from the **Empty** project template, the *CMS Advanced Development* course is designed with stand-alone modules so that they can be completed in any order. Every module has hands-on exercises that can be completed by starting with a freshly created and updated **Alloy (MVC)** site.

- All exercises are dependent on the completion of **Exercise 0.1**, which sets up an **Alloy (MVC)** site with updated NuGet packages and updated CMS database schema.
- Some exercises require the Northwind sample database, created in **Exercise 0.2**

We picked the Alloy reference site as a starting point because

- It is built-in with the Episerver CMS Visual Studio extension, It is quick to set up with some sample content, It is small enough to understand, and familiar to many Episerver developers, and it shows some good practices. Episerver Alloy MVC: http://www.awareweb.com/awareblog/4-17-17-episerver_10_alloy_mvc

Episerver 9

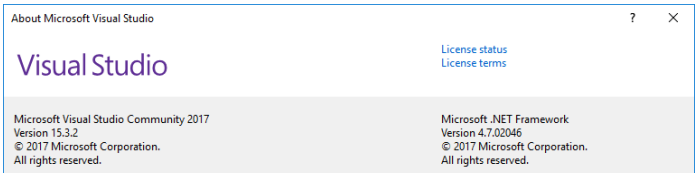
Introduction – About the course

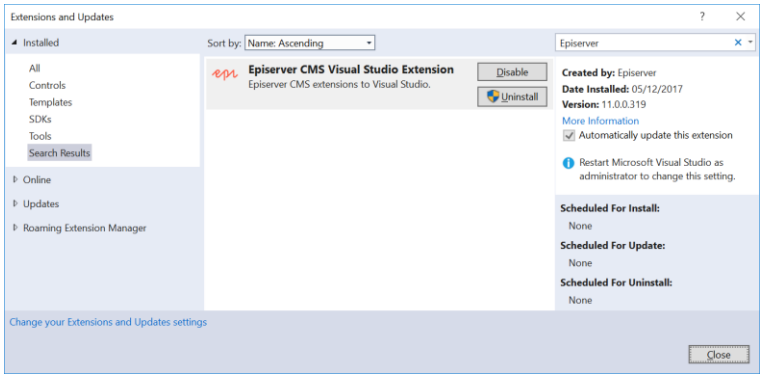
Software requirements

- Microsoft Visual Studio 2017 version 15.3 or later (with latest updates)
- Episerver CMS Visual Studio Extension 11.1.0.326 or later (includes CMS 11.3.0)

Links to older CMS extension versions:

http://world.episerver.com/download/Items/Episerver-CMS/visual-studio-cms-extensions/





Episerver 10

Introduction – Getting more information

Knowing where to get help

Episerver World is where you go to read the documentation for CMS developers:

- http://world.episerver.com/cms

You can ask questions and make feature requests in the forums:

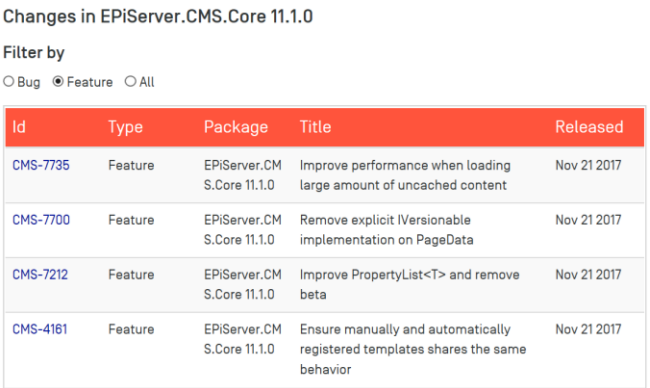
- http://world.episerver.com/forum
- http://world.episerver.com/forum/developer-forum/Feature-requests/

You can record a support ticket:

- http://world.episerver.com/support

You can find a list of fixed bugs and new features about a specific release:

- http://world.episerver.com/releases and http://world.episerver.com/documentation/Release-Notes/



Id	Type	Package	Title	Released
CMS-7735	Feature	EPiServer.CMS.Core 11.1.0	Improve performance when loading large amount of uncached content	Nov 21 2017
CMS-7700	Feature	EPiServer.CMS.Core 11.1.0	Remove explicit IVersionable implementation on PageData	Nov 21 2017
CMS-7212	Feature	EPiServer.CMS.Core 11.1.0	Improve PropertyList<T> and remove beta	Nov 21 2017
CMS-4161	Feature	EPiServer.CMS.Core 11.1.0	Ensure manually and automatically registered templates shares the same behavior	Nov 21 2017

CMS 11.1 or later requires
- Commerce 11.5 or later
- Forms 4.9 or later
- Search 9.0.1 or later
- Find 12.7 or later
- Social Reach 2.3 or later
- Google Analytics 2.0 or later
- Languages 3.1 or later

Episerver 12

Introduction – Getting more information

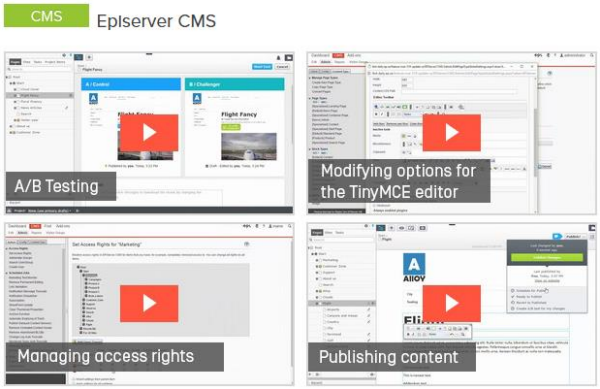
Episerver videos

For editors and administrators:

http://webhelp.episerver.com/latest/_online-only-topics/videos.htm

For developers:

http://world.episerver.com/documentation/videos/



Episerver 13

Introduction – Getting more information

Episerver product team blogs

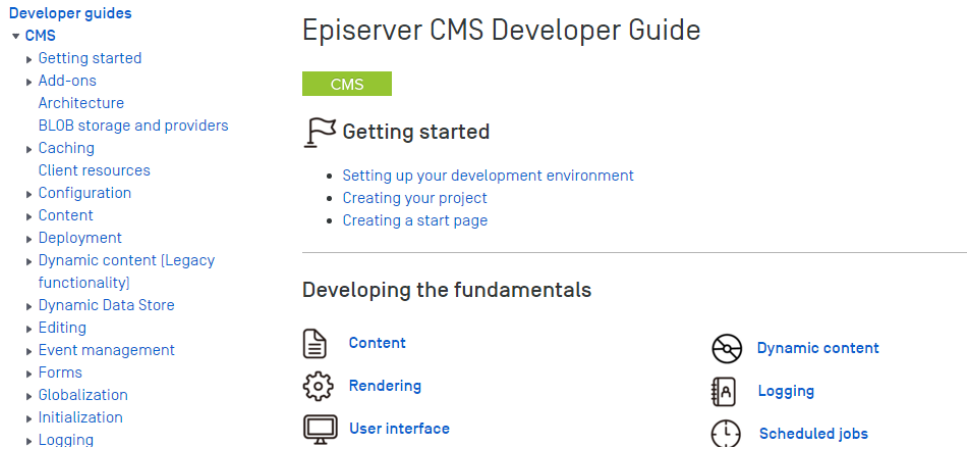
- The Cloud Team blog
  http://world.episerver.com/product-blogs/the-cloud-blog/
- The CMS Team blog
  http://world.episerver.com/product-blogs/the-cms-blog/
- The Commerce Team blog
  http://world.episerver.com/product-blogs/the-commerce-blog/
- The Find Team blog
  http://world.episerver.com/product-blogs/the-find-blog/
- The Campaign Team blog
  http://world.episerver.com/product-blogs/the-campaign-blog/
- The Social Team blog
  http://world.episerver.com/product-blogs/the-social-blog/
- The Personalization Team blog
  http://world.episerver.com/product-blogs/the-personalization-blog/

Episerver 14

Introduction – Getting more information

CMS documentation

http://world.episerver.com/documentation/developer-guides/CMS/



Episerver

Introduction – Getting more information

Partners and EMVPs with useful blogs about Episerver

- **Ted & Gustaf**: Episerver Premium Partner. https://tedgustaf.com/blog/
- **Alf Nilsson talks**: Babble about EPiServer, and other development. https://talk.alfnilsson.se/
- **David Knipe**: former EMVP, now Principal Solution Architect, Episerver UK. https://www.david-tec.com/
- **Deane Barker**: Chief Strategy Officer, founding partner at Blend Interactive. http://gadgetopia.com/
- **Fredrik Haglund**: independent consultant and Episerver trainer. http://blog.fredrikhaglund.se/
- **Grzegorz Wiecheć**: MCPD and Certified EPiServer developer. https://gregwiechec.com/
- **Aria Zanganeh**: software developer who is passion about technology. http://azanganeh.com/
- **Māris Krivtežs**: EPiServer and front-end development at Geta. http://marisks.net/
- **Wałdis Iljuczonok**: http://blog.tech-fellow.net/
- **Episerver Fellow**: http://fellow.aagaardrasmussen.dk/
- **Jon D. Jones**: UK consultant who regularly blogs about CMSes. http://jondjones.com/

Episerver 16

Introduction – Product history

Episerver CMS – the first decade

Development started 1996 by ElektroPost on the iServer product

- **Version 1**, release 1997
- **Version 2**, release 1998
  - Inline rich text editor
- **Version 3**, release 2000
  - Frameless
  - Customizable
- **Version 4**, ASP.NET 1.0, release 2002
- **Version 4.6**, support for both ASP.NET 1.1 and 2.0, released 2005

Episerver 18

Introduction – Product history

Episerver CMS 5 – 2005 to 2008

- **Episerver CMS 5**, release 2007
  - ASP.NET 3.0
  - Considerable changes made to the API and available tools
    - Visual Studio Integration
  - Over 3000 sold licenses

- **Episerver CMS 5, R2**, release 2008
  - Page Providers
  - Dynamic Content
  - Improved Image Editor, …

Episerver 19

Introduction – Product history

Episerver CMS 6 – 2010 to 2011

- **Episerver CMS 6 R1** released in March 2010
  - Drag-and-drop functionality in Edit Mode and Admin Mode
  - Compare page versions (color coding, property and side-by-side)
  - New Editor (TinyMCE)
  - Dynamic Data Store (DDS)
- **Episerver CMS 6 R2** released in April 2011
  - Visitor Groups and Personalization
  - Autosave for pages in Edit Mode
  - Container Pages

Introduction – Product history

Episerver CMS 7 – 2012

- Released in November 2012
  - Blocks and Content Areas
  - Strongly typed content
  - Separation of data and rendering; multiple templates per content type, rendered based on context
  - Support for MVC 4
  - Testability: Abstractions and interfaces supporting IoC
  - Episerver Search integrated in the CMS product
  - Add-on Store
  - New UI built with DOJO: “Edit Mode” becomes “Edit View”
  - “One common draft” for Editors

Introduction – Product history

Episerver CMS 7.1 – 2013

- Released in April 2013
  - Settings for pages and blocks visible in the on-page view
  - Personalization of blocks
  - Improved workflows – the four WWF workflows now available
  - Multivariate testing of blocks
  - Improved language handling of globalized websites
    - Block content now translated in the same way as page content
    - Language selector added to the "View Settings" menu
  - JavaScript source code package for Shell.UI and Episerver CMS available as an add-on

http://world.episerver.com/Documentation/Items/Release-Notes/Episerver-CMS/Episerver-7/Release-Notes--Episerver-7-1-CMS/

Episerver 22

Introduction – Product history

Episerver CMS 7.5 – 2013

- Released in December 2013
- On-page-editing for XHTML and long string - no flyout window
- Introducing local folders: For All Sites, For This Site, For This Page / For This Block
- Adding and editing link collections
- Media files (images, documents, video) and products in the Commerce catalog are now handled as content types in CMS
- XCOPY deployment
- Cloud support – for running and deploying CMS in Azure and Amazon
- Webmasters can launch new sites directly from CMS Admin

http://world.episerver.com/Documentation/Items/Release-Notes/Release-notes---Episerver-75/

Episerver 23

Introduction – Product history

Episerver CMS – 2015

- **Version 8, 23 February 2015**
  - Multi-publish and preview
  - Support for canonical URLs
  - Better ways to organize content types and properties
  - Performance improvements.
  - http://world.episerver.com/articles/Items/new-release-of-episerver-cms-with-enhanced-preview-and-improved-seo/
- **Version 9, 22 September 2015**
  - http://world.episerver.com/releases/episerver--update-81/

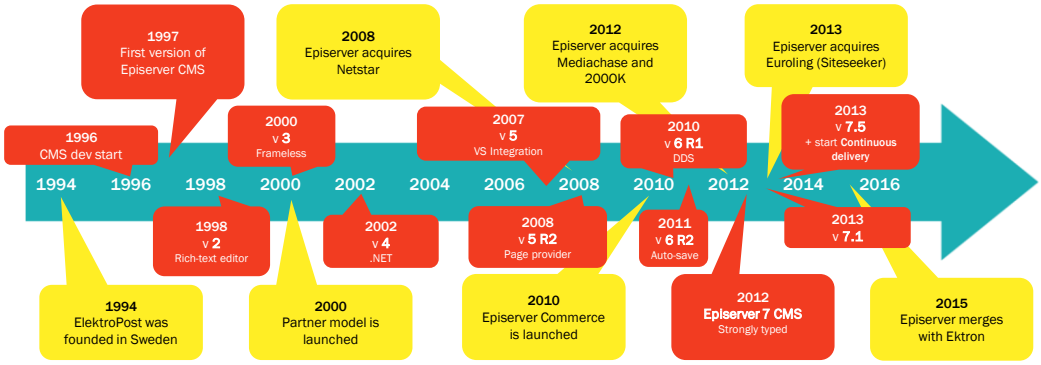
Introduction – Product history

Episerver CMS – 2016 to present

- **Version 10, 24 October 2016**
  - Content Approvals
  - A/B Testing
  - http://world.episerver.com/releases/episerver--update-134/
  - http://world.episerver.com/documentation/upgrading/Episerver-CMS/10/breaking-changes-cms-10/
- **Version 11, 21 November 2017**
  - .NET Standard 2.0-compliant core packages
  - Split packages: XForms, Dynamic Content, TinyMCE, StructureMap, Episerver Search
  - http://world.episerver.com/releases/episerver--update-189/
  - http://world.episerver.com/documentation/upgrading/Episerver-CMS/cms-11/breaking-changes-cms-11/
  - http://world.episerver.com/documentation/upgrading/Episerver-CMS/cms-11/new-nuget-packages/

Introduction – Product history

Key milestones for Episerver CMS



Introduction – Episerver CMS 11 breaking changes

Breaking changes in Episerver CMS 11

Episerver plans for one breaking change release per year.

http://world.episerver.com/documentation/upgrading/Episerver-CMS/cms-11/breaking-changes-cms-11/

If you stay up-to-date with continuous releases, and make note of our warnings about APIs that will become obsolete, then a major version number update often only requires a project re-compile, after reviewing potential breaking changes.

One of Episerver's continuing goals is to slowly obsolete non-.NET Standard 2.0-compatible APIs.

- For example, the `CreatePropertyControl` method in `PropertyData` has been removed since it has a dependency on `System.Web.UI.Control` which is part of the legacy technology ASP.NET Web Forms.
- Target .NET Framework 4.6.1 in order to be compliant with .NET Standard 2.0.

Improve performance when loading large amount of uncached content

http://world.episerver.com/documentation/Release-Notes/ReleaseNote/?releaseNoteId=CMS-7735

Headless E-Commerce, Non-Starter or the Next Big Thing?
https://www.websitemagazine.com/blog/headless-e-commerce-non-starter-or-the-next-big-thing

Splitting of NuGet packages for .NET Standard 2.0

The goal was to split our NuGet packages into .NET Standard 2.0-compatible and non-.NET Standard 2.0-compatible packages.

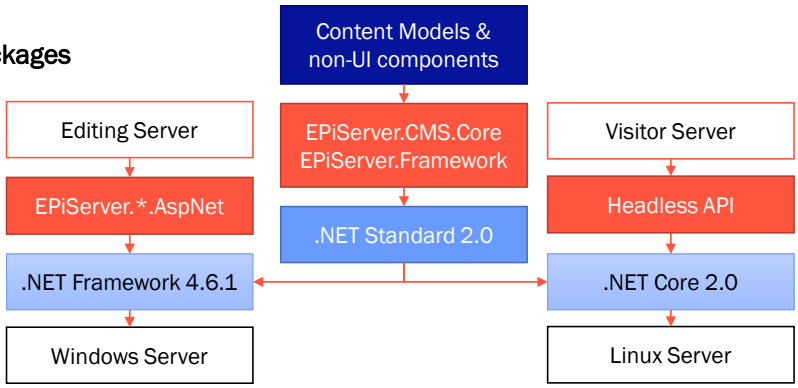
.NET Standard 2.0-compatible packages

- `EPiServer.CMS.Core`
- `EPiServer.Framework`

Non-compatible packages

- `EPiServer.CMS.AspNet`
- `EPiServer.Framework.AspNet`

Some Web.config entries now refer to these new packages, for example, to configure the localization provider.



Episerver 29

TinyMCE editor, Dynamic Content, and XForms NuGet packages

The **TinyMCE editor** and related plugin configuration features has been moved into a separate NuGet package as an add-on with its own versioning number and breaking changes. This is to allow us to have a release cycle for TinyMCE which is decoupled from the CMS UI release cycle.

The legacy features **Dynamic Content** and **XForms** have been removed from the platform and moved into separate NuGet packages as add-ons:

- `EPiServer.DynamicContent`
- `EPiServer.XForms`

```
Install-Package -ProjectName AlloyDemo EPiServer.XForms
```

These packages now have their own version number and breaking changes, and will be updated less frequently.

As the platform progresses these features will become more limited over time, so we recommend migrating from Dynamic Content to **Blocks**, and from XForms to **Episerver Forms** as soon as possible.

Episerver 30

StructureMap NuGet package

A new NuGet package, EPiServer.ServiceLocation.StructureMap, supports:

- Existing signed StructureMap 3, and
- New unsigned StructureMap 4.

The package only contains the integration, so it has NuGet dependencies on the official StructureMap packages.

Moving the dependency to a NuGet package is the same approach we have for logging, where we have abstractions in the platform and an integration in a separate NuGet package. This will allow us to more easily swap to the dependency injection system that is shipped with .NET Core, if we choose to do so in the future.

Exercises for Introduction

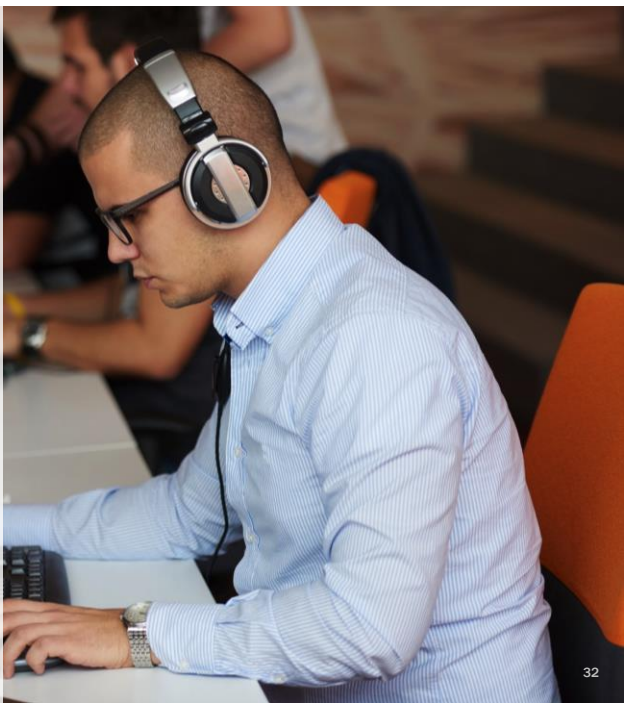
0.1 Setting up the Alloy (MVC) site

In this exercise, you will set up an Alloy (MVC) sample site ready to extend.

0.2 Setting up the Northwind database

Exercises C2, C4, and D4 use external database.

Prerequisites: Microsoft Visual Studio 2015 or 2017 with Episerver CMS Visual Studio Extension 11.1.0.326 or later.



Episerver CMS Advanced Development

Module A
Reviewing Episerver CMS Fundamentals

Map Episerver's capabilities to project requirements and get a good overview of what needs to be developed.

Topics

- ASP.NET MVC architecture and conventions
- Designing an Episerver web site
- Localizing for Editors and Visitors
- Architecture
- Dependency Injection (DI)
- Initialization
- Logging
- Security
- Caching
- Good practice

HTTP methods and status codes

"Web" programmers are HTTP programmers. How much do you know about HTTP?

What are some of the common HTTP methods/verbs?

- Most common: GET, POST, PUT, DELETE
- Others include: PATCH, OPTIONS, CONNECT, HEAD

What are some of the common HTTP status codes and what do they mean?

- **1xx** means Information. **101**: Switching protocols, e.g. from HTTP to Web Sockets.
- **2xx** means Success. **200**: OK, **201**: Created, **204**: No content.
- **3xx** means Redirect. **301**: Moved permanently, **302**: Moved temporarily.
- **4xx** means Client error. **400**: Bad request, **401**: Unauthorized, **404**: Not found.
- **5xx** means Server error. **500**: Server exception thrown.

ASP.NET intrinsic objects

What are some of the ASP.NET intrinsic objects and what do they do?

- **HttpContext**: encapsulates all other intrinsic objects related to the current request.
- **Request**: the incoming request, e.g. cookies, headers, user agent.
- **Response**: the outgoing response, e.g. cookies, headers.
- **Application**: a dictionary shared between all users, e.g. list of countries. (Use Cache instead.)
- **Cache**: a dictionary shared between all users, e.g. list of countries.
- **Session**: a dictionary for a user session, e.g. shopping cart.
- **Items**: a dictionary for the current HTTP context.
- **Server**: the web server's properties and methods.
- **User**: the logged in user.

ASP.NET MVC conventions

- What routes are configured by default with ASP.NET MVC? What are the segments used for? Which segments are mandatory?
  - `http://www.example.com/{controller}/{action}/{id}`
  - All segments are optional. {controller} defaults to **Home** and {action} defaults to **Index**.
- If the **{controller}** segment has a value of **home**, what is the name of the class that ASP.NET MVC instantiates? Which method will it execute?
  - The class name must be **HomeController** and by default it will execute a method named **Index**.
- If you call the **View** method inside a controller's action method, what path(s) does it search to find the view? What file extensions are searched for?
  - Paths: `~/Views/{controller}/{action}`, `~Views/Shared/{action}`
  - File extensions: .aspx, .ascx, .cshtml, .vbhtml

Module A – Reviewing Episerver CMS Fundamentals – ASP.NET MVC architecture and conventions

Episerver conventions

Episerver uses some of the ASP.NET MVC conventions, e.g.,

- **Index** is the default action method name, and
- the **View()** method of a Controller looks in the same paths for views. (Note: the Alloy sample site adds two paths: `~/Views/Shared/Blocks` and `~/Views/Shared/PagePartials`)

…but Episerver changes other conventions, e.g. the **Default** route is replaced with multiple custom Episerver routes.

Do you know what segments Episerver configures? Which of them are optional?

- `http://www.example.com/{language}/{page node segments}/{partial route}/{action}/`
- All segments are optional. If no page node segments are specified the Start Page is shown using the default language and the Index action method is called on the controller that provides the content template, or a 404 is shown.

Model binding in ASP.NET MVC

What is a model binder?

- Converts data in an HTTP request into validated action method parameter values.

How many model binders are included in ASP.NET MVC?

- Five. `DefaultModelBinder`, and four specialised ones.

How can you check that a parameter value passes any validation rules you have applied to the model?

- Inside the action method, check the `ModelState.IsValid` property.

ASP.NET MVC - The Features and Foibles of ASP.NET MVC Model Binding
https://msdn.microsoft.com/en-us/magazine/hh781022.aspx

6 Tips for ASP.NET MVC Model Binding:
http://odetocode.com/blogs/scott/archive/2009/04/27/6-tips-for-asp-net-mvc-model-binding.aspx

Module A – Reviewing Episerver CMS Fundamentals – ASP.NET MVC architecture and conventions

Default model binding in ASP.NET MVC

What data types can the `DefaultModelBinder` handle?

- Simple types, e.g., `int`, `double`, `string`, and so on.
- Complex model types, e.g. `Person`, `Animal`, and so on.
- Collection types, e.g. arrays, `IEnumerable<T>`, `List<T>`, and so on.

What algorithm does the DefaultModelBinder use to map set parameters for the action method?

- It compares the method parameter names with the parameter names from the following parts of the incoming HTTP request, until all parameters are populated, otherwise it returns a 404:
  1. Form
  2. Route
  3. QueryString
  4. File

```
// DefaultModelBinder will set the parameters automatically
public ActionResult Index(Person person, string color, int id)
```


Model binding in Episerver

Do you know what extra functionality Episerver's model binder provides?

- It sets action method parameters with the following special names (case-insensitive)
  - **currentcontent**
  - **currentpage**
  - **currentblock**

```
// Episerver's model binder will set the parameter automatically
public ActionResult Index(ProductPage currentPage)
```

Episerver

Module A – Reviewing Episerver CMS Fundamentals – ASP.NET MVC architecture and conventions

Model validation attributes

What are some of Microsoft's attributes that can decorate your model classes for validation?

- **[Required]**: when creating a new page or block, this will force editor to populate this property.
- **[StringLength(50)]**: restricts length of a string to a maximum (50) and a minimum (optional).
- **[Range(18, 65)]**: restricts an integer value to a minimum (18) and a maximum (65).
- **[RegularExpression("[A-Z]+")]**: automatically includes ^ and $.
- **[DataType(DataType.Date)]**: CreditCard, DateTime, Date, Time, Duration, PhoneNumber, Currency, Text, EmailAddress, Password, Url, ImageUrl
- **[CreditCard]** and **[EmailAddress]**: same as DataType with CreditCard or EmailAddress.
- **[CustomValidation(typeof(MyClass), "MyMethod")]**: define a static class with a static method.

What is the recommended way to provide a custom validation attribute?

- Create a class that inherits from `ValidationAttribute`.

Episerver

Model non-validation attributes

What are some of Microsoft's attributes that can decorate your model classes?

- **[ScaffoldColumn(false)]**: does not output the property.
- **[Hidden(false)]**: outputs the property using `<input type="hidden" />`
- **[Ignore]**: does not store the property when serialized (e.g. for storage).
- **[DisplayFormat]**: custom the output using standard format string e.g. "dddd, d MMMM yyyy".
- **[Display]**: customize the label, description, sort order, and so on.
- **[DisplayName]** (deprecated): similar to Display but it cannot be localized.

Episerver reuses Microsoft's validation and non-validation model attributes, in particular, **[Display]**, to control **All Properties** view when editing content.

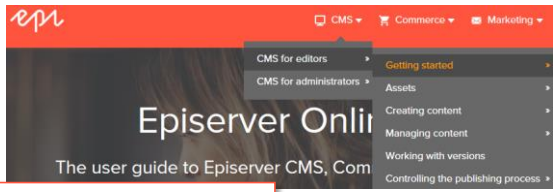
Quick Demonstration
- Visitor navigates Alloy products and searches for 'team'.
- Editor changes Alloy product name and main body.
- Admin runs scheduled job and assigns access rights.

Why use a CMS?

- What are the benefits of using a Content Management System? Why not just build a site with Microsoft ASP.NET MVC?

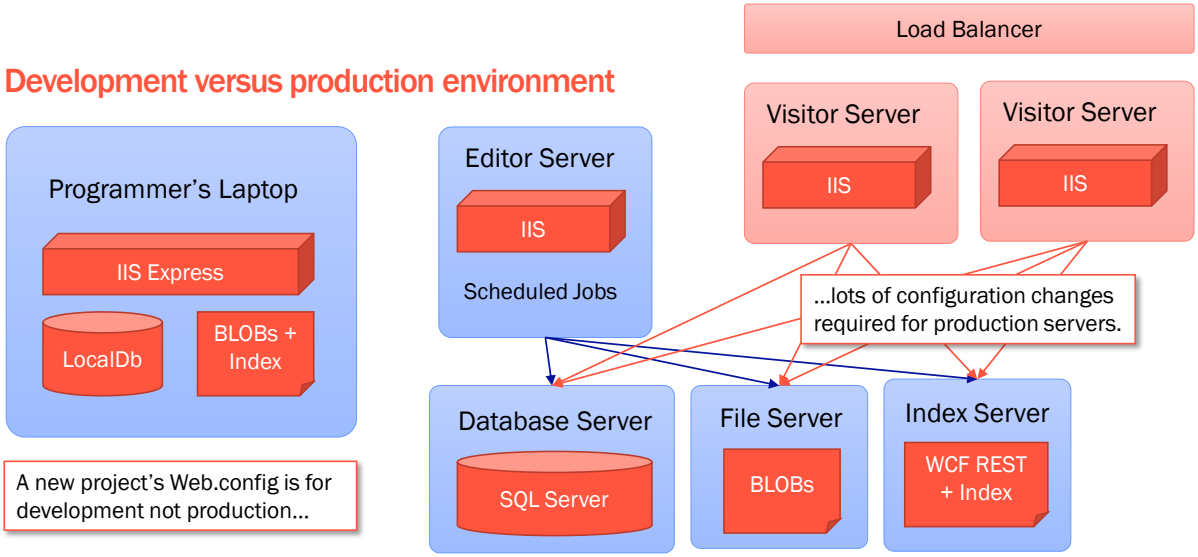
1. Easy for non-technical people to create professional well-structured content.
2. Flexible access control lists for applying permissions to content.
3. Localize content into multiple human languages.
4. Control publishing workflow and multiple versions.

- Developers should know about CMS features for editors and administrators, so read the **User Guides:** http://world.episerver.com/documentation/Items/user-guides/



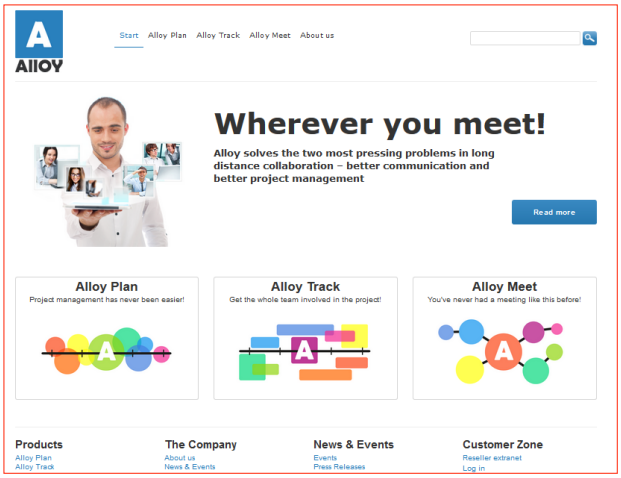
Homework, Learn basic editing
http://world.episerver.com/documentation/developer-guides/CMS/getting-started/learn-basic-editing/

Development versus production environment



A new project's Web.config is for development not production...

Identify parts



- Logo
- Top menu
- Search field
- Large teaser
- Small teasers
- Page footer

Using blocks

- What is the difference between a block that is used as a property and a block that is shared?
- `TeaserBlock`: Same teaser is used on several pages.
  - Shared block, added to `ContentArea`

  http://world.episerver.com/documentation/developer-guides/CMS/Content/Block-types-and-templates/
- `LogoBlock`: One block used on every page
  - Block used as property

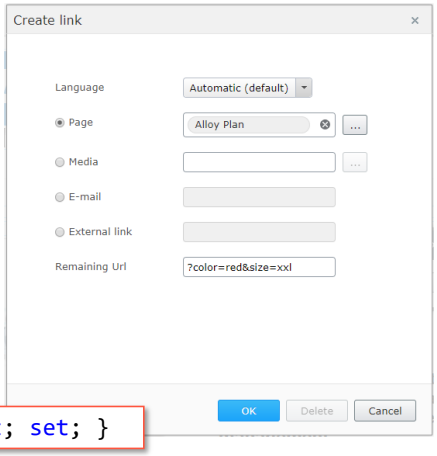
  http://world.episerver.com/documentation/developer-guides/CMS/Content/Properties/using-a-block-as-a-property/
- What is the difference between **For All Sites**, **For This Site**, and **For This Page** / **Block**?
- **For This Site** is not available by default. How is it activated?
- Where should site settings be stored?

Linking to other content

- Both TeaserBlock and LogoBlock have **Image** properties. Why does one use **ContentReference** and the other use **Url**? What's the difference?

  http://world.episerver.com/documentation/developer-guides/CMS/Content/Links/Linking-to-other-content/
- TeaserBlock
  - Heading: string
  - Text: string
  - **Image: ContentReference**
- LogoBlock
  - Title text: string
  - **Image: Url**

Query strings with Url and ContentReference



Url and items in a **LinkItemCollection** can link to internal and external items.

```
public virtual Url MyUrl { get; set; }
```

- If they link to a page or media, you can add query string parameters using **Remaining Url**, as shown in the screenshot:

How could you add a query string value to a **ContentReference**?

```
public virtual ContentReference MyContentReference { get; set; }
```

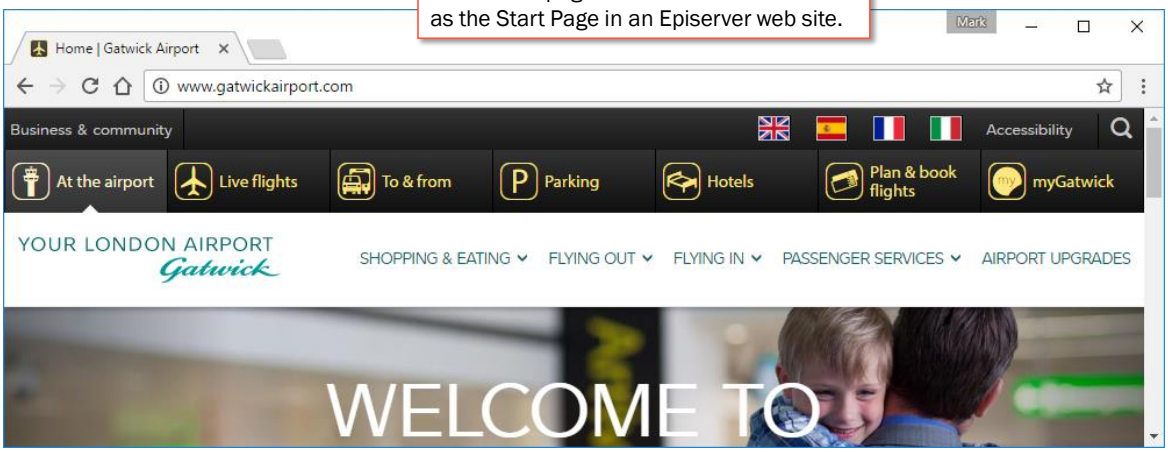
```
<li @Html.EditAttributes(m => m.CurrentPage.MyContentReference)>
  <a href="@Url.ContentUrl(Model.CurrentPage.MyContentReference)?color=red&size=xxl">
    My Link</a></li>
```

Episerver 51

Module A – Reviewing Episerver CMS Fundamentals – Designing an Episerver web site

Gatwick Airport: Start page

The default page shown is the one selected as the Start Page in an Episerver web site.

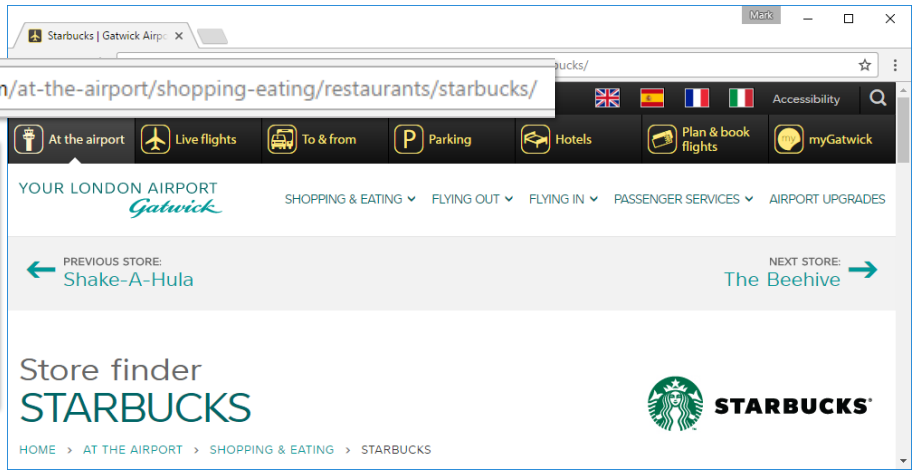


Episerver

Module A – Reviewing Episerver CMS Fundamentals – Designing an Episerver web site

Gatwick Airport: "greedy" page nodes

The Gatwick site does not use the `{culture}` segment. The `{page node segments}` are "greedy", meaning they "eat" every segment that has a matching page in the tree, so they can be any depth.



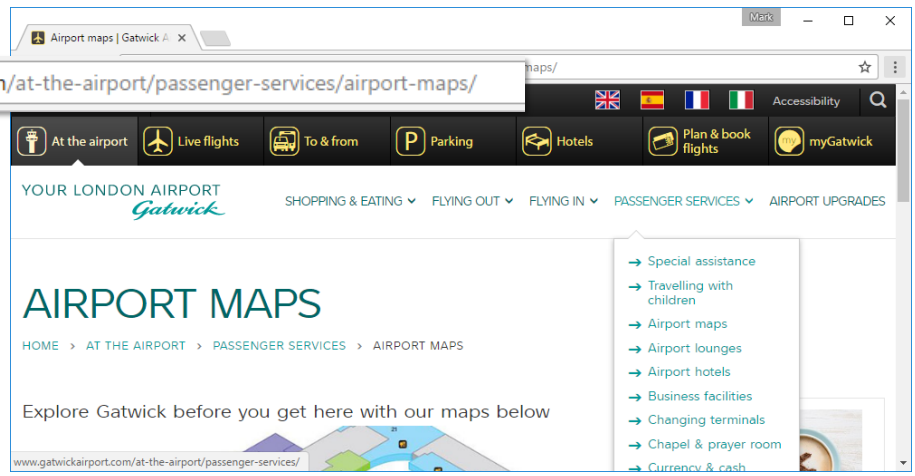
Episerver

Module A – Reviewing Episerver CMS Fundamentals – Designing an Episerver web site

Gatwick Airport: navigating with menus and breadcrumbs

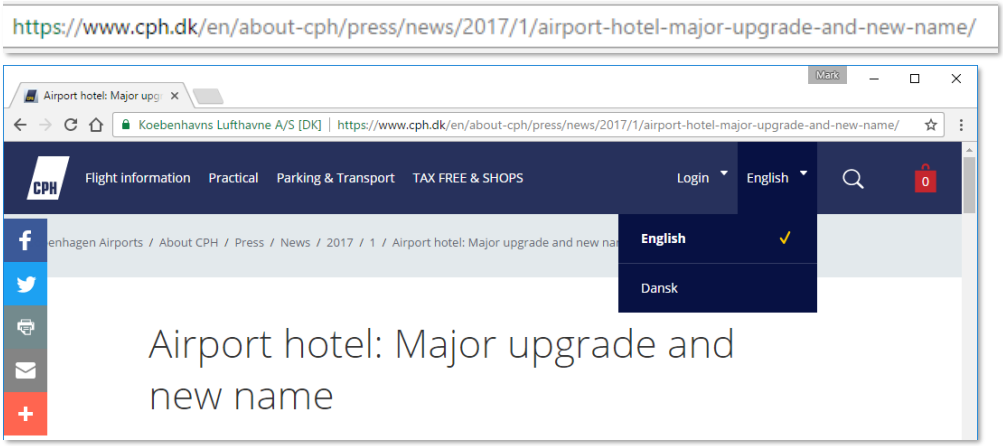
The Gatwick site has navigation with:

- Three levels of menu, each with different styling.
- A breadcrumb to navigate back up to ancestor pages.



Episerver

Copenhagen Airport: a news page with localization and year/month page containers



Episerver

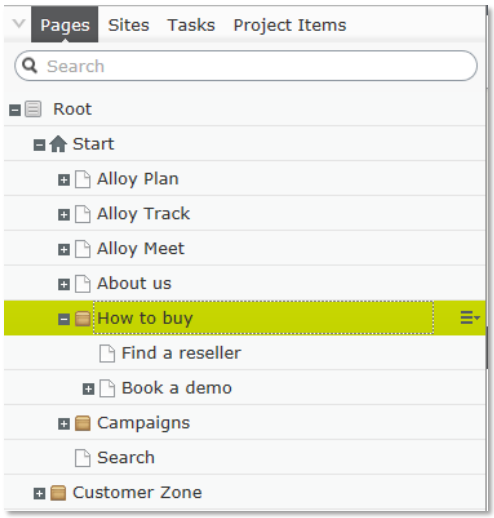
Module A – Reviewing Episerver CMS Fundamentals – Designing an Episerver web site

Container pages and data pages

All pages can be a parent of others pages, so in that sense all pages are “container pages”. When Episerver developers talk about **container pages**, they usually mean a page without a template, meaning that it cannot be rendered to a visitor. They are designed only to “contain” children.

Examples in Alloy are the **How to buy** and **Campaigns** pages, or the **Contacts** page underneath **About us**.

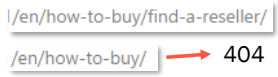
A similar concept are **data pages** (template-less leaf pages). These store data that “belongs” to their parent page but render only as part of the parent. They do not render as full pages themselves. For example, FAQItems.



Episerver

56

Container pages do not have templates

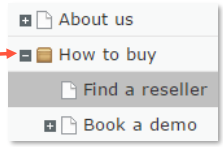


First, know that some developers have a strong opinion that container pages are bad, because the default URL behaviour would be to show a 404 for the container page URL.
https://www.epinova.no/en/blog/container-pages-and-why-you-shouldnt-use-them/

Second, if you want to use them, simply create a page type *without* a page template. For example, a class named `ContainerPage` that derives from `PageData`, with no `PageController<ContainerPage>`.

Optionally, create an UI descriptor to change the icon shown in the Navigation pane page tree:

```
[UIDescriptorRegistration]
public class ContainerPageUIDescriptor : UIDescriptor<ContainerPage>
{
    public ContainerPageUIDescriptor()
        : base(ContentTypeCssClassNames.Container)
    {
        DefaultView = CmsViewNames.AllPropertiesView;
    }
}
```



Common Episerver design patterns and conventions

When you create your own site, use **Site** as a prefix for types that extend the built-in Episerver API types. For example:

- Episerver has content types `PageData`, `BlockData`, and `MediaData`. Create derived types named `SitePageData` and so on with properties that will be common to all pages and so on in your site.

Microsoft has a convention of using **Base** as a suffix for abstract classes that they expect other developers to derive from. You can do the same. For example:

- Episerver has a type named `PageController<T>`. Create a derived type named `PageControllerBase<T>` with methods that will be common to all page templates on your site.

An alternative to having a base controller is to have a separate non-Episerver MVC controller for common action methods. For example: **SiteController** with **LogOff** action method.

Setting icons for content types

Episerver has an attribute named `ImageUrlAttribute`. Create a derived type named `SiteImageUrlAttribute` that has a default constructor that sets the path to a default image file:

```
public class SiteImageUrlAttribute : ImageUrlAttribute
{
    public SiteImageUrlAttribute()
        : base("~/Static/contenticons/epi-edu-icon.jpg") { }

    public SiteImageUrlAttribute(string path)
        : base(path) { }
}
```

Apply this attribute to your content type classes:

```
[SiteImageUrl]
public class StartPage : PageData
```


Site group and tab names

It is good practice to avoid “magic” strings in your code.

ContentType and **Display** attributes both allow you to specify a **GroupName** as a string value.

You should define a static class with string constants instead of just setting literal string values.

The recommendation is to define your own **SiteTabNames** and **SiteGroupNames** static classes.

```
namespace AlloyTraining
{
    public static class SiteGroupNames
    {
        // this will be used for Start and Search pages
        public const string Specialized = "Specialized";
        // this will be used for all other pages
        public const string Common = "Common";
```


Using a view model

An alternative to having a view model is to add **[Ignore]** to properties in the page type. This prevents them from being stored in the CMS. But beware of caching!
http://blog.q1.se/2016/03/08/rule-of-thumb-never-have-ignore-properties-in-a-contenttypemodel/

Frequently you need more than just the Page object in your View, so it is common to create a ViewModel class. Create an interface and use inheritance so that your strongly-typed models can be passed to your layouts as well as the views.

Interface and base class:

```
public interface IPageViewModel<out T> where T : SitePageData
{
    T CurrentPage { get; }
}
```

```
public class PageViewModel<T> : IPageViewModel<T> where T : SitePageData
```

_Layout.cshtml:

```
@model IPageViewModel<SitePageData>
```

StartPage\Index.cshtml:

```
@model PageViewModel<AlloyDemo.Models.Pages.StartPage>
```


Data that is not content

Can we find anything in Alloy other than editorial content e.g. pages, blocks, media assets?

Where would you store data that is not content?

- Use **Dynamic Data Store (DDS)** to store simple custom data, e.g. for add-ons
  - Example: XForm and Form submissions, Visitor Groups definitions and statistics.
  - DDS has no user interface for editing. Although a custom user interface to DDS could be built, data that needs to be updated by content editors is often not suited for DDS.
  - Consider using "data pages" instead of DDS.
- Use **RDBMS** or **NoSQL** stores for more complex or relational data.

You will learn more in *Module C: Integrating Data and Forms*

Make the life easier for the editors

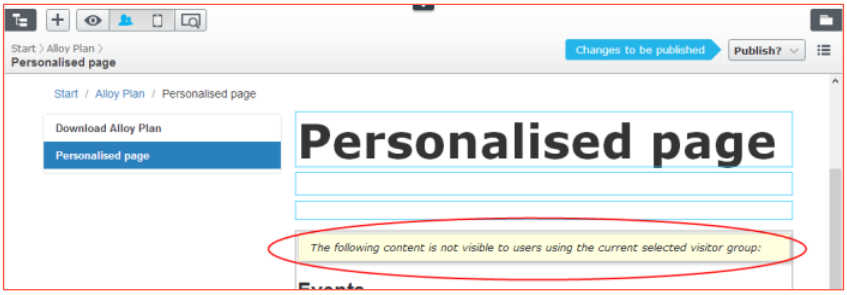
The site is custom built for our visitors, **but don't forget the editors**! The editors or administrators might need some custom tools to make their work life easier.

- Control the page tree structure by applying [`AvailableContentTypes`] to page types.
- Customize the **New Page** and **New Block** user interface by applying [`ContentType`] and [`ImageUrl`] to content types to provide a name, description, and icon.
- Use localization to support multiple languages for content type names and descriptions, and property names and descriptions.
- Create plug-ins to extend the user interface, e.g. Versions gadget.
  You will learn more in *Module F: Extending with Plug-ins and Add-ons*

Module A – Reviewing Episerver CMS Fundamentals – Designing an Episerver web site

Make the life easier for the editors

- Create custom validation, for example, implement `IValidate<T>` for content types.
- Custom properties: if the functionality you need is not covered by `UIHint` and built-in property types you can add your own custom property types.
- Create on-page editing instructions, by using `PageEditing.PageIsInEditMode`
- You will learn more in *Module D: Customizing Properties for Editors*



Content areas and blocks on a globalized site

- Language dependent content areas
  vs.
- Language independent content areas.

- What does the `[CultureSpecific]` attribute do when applied to:
  - A property of type **string** or **XhtmlString**?
  - A property of type **ContentArea**?

Localization service configuration

Even if you don't intend to localize into more than one language, you should still learn how to use localization resource files because they override code and administrator settings.

The default for fallback behavior is echo without missing message.

```
<episerver.framework>
  <localization fallbackBehavior="Echo, MissingMessage, FallbackCulture"
                fallbackCulture="en">
    <providers>
      <add virtualPath="~/Resources/LanguageFiles" name="languageFiles"
           type="EPiServer.Framework.Localization.XmlResources
                 .FileXmlLocalizationProvider, EPiServer.Framework.AspNet" />
    </providers>
  </localization>
```

Breaking change in CMS 11: Assembly name has changed.

Localization service GetString method

If the following XML file exists in the specified resource folder, then the LocalizationService's **GetString** method would work like this:

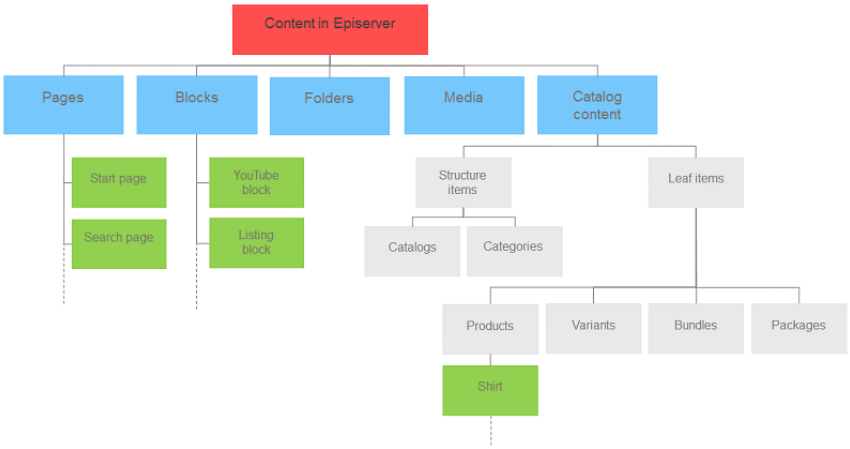
```
<?xml version="1.0" encoding="utf-8" ?>
<languages>
  <language name="English" id="en">
    <buttonSave>Save</buttonSave>
    <menuFile>
      <menuNew>New</menuNew>
      <menuOpen>Open</menuOpen>
    </menuFile>
```

```
private readonly LocalizationService localizer;
```

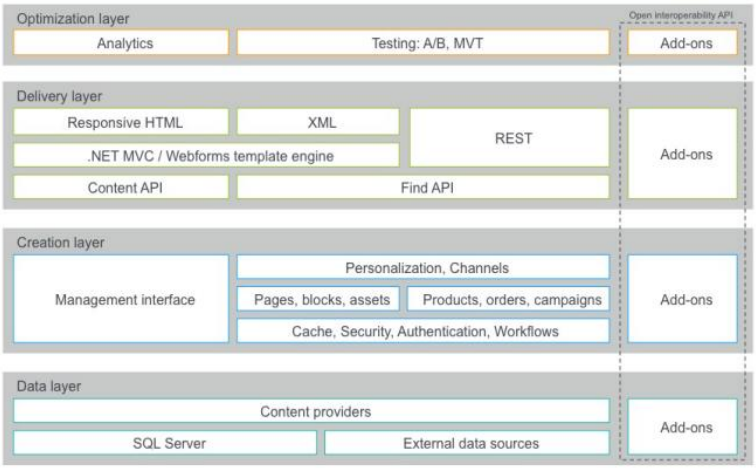
```
var value1 = localizer.GetString("buttonSave");          // Save
var value2 = localizer.GetString("/buttonSave");         // Save
var value3 = localizer.GetString("menuOpen");            // menuOpen
var value4 = localizer.GetString("menuFile/menuOpen");   // menuFile/menuOpen
var value5 = localizer.GetString("/menuFile/menuOpen"); // Open
var value6 = localizer.GetString("/buttonClose");
    // [Missing text '/buttonClose' for 'English']
```

Recommendation: always use / as prefix, followed by element names separated by /

Episerver CMS and Commerce content



Episerver technical overview – diagram



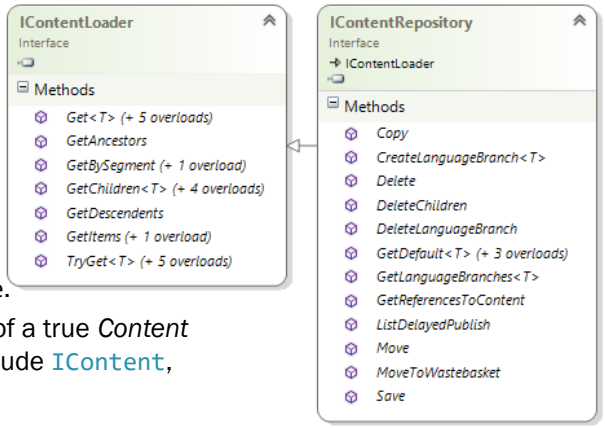
Module A – Reviewing Episerver CMS Fundamentals – Architecture

Episerver technical overview – description

- The **optimization layer** provides various options for measuring, analyzing and optimizing the performance of website content, for instance conversion rates for a campaign landing page.
- The **delivery layer** holds the presentation parts including support for responsive design and templates based on WebForms or MVC, and support enabling the building of advanced search and filtering features.
- The **creation layer** is where content, for instance pages and blocks or e-commerce content such as products and orders, is managed. Content can be personalized, and can also be part of an approval workflow with authorization applied.
- The **data layer** provides the information, which can originate from the database, from one or more content providers or even from external data sources integrated with Episerver.

http://world.episerver.com/documentation/developer-guides/CMS/architecture/

API access to the Episerver CMS database



In pre-7 versions, Episerver CMS was more of a *Page* Management System. Types and members included `PageData`, `PageReference`, and `PageName`.

Since version 7, it has been refactored to be more of a true *Content* Management System. New types and members include `IContent`, `ContentData`, `ContentReference`, and `Name`.

The old `DataFactory` has grown too big so should be avoided for performance and unit testing.

Services that implement smaller sets of functions should be used instead. The two most common are:

- `IContentLoader`: read-only access to Epi database.
- `IContentRepository`: full CRUD access to Epi database.

Other Episerver interfaces and types for services

`IContentTypeRepository`: CRUD with content types in Episerver database.

`IContentVersionRepository`: list and delete content versions in Episerver database.

`IContentEvents`: listen for events during CMS lifecycle, e.g. publishing a page.

`IPageCriteriaQueryService`: search for content in Episerver database (not indexed).

`UrlResolver`: convert content reference to public URL and other tasks.

`LocalizationService`: read localized strings from XML files (or custom provider).

`DisplayOptions`: allow editor to customize which template is used for a block.

Dependency injection

http://world.episerver.com/documentation/developer-guides/CMS/initialization/dependency-injection/

How to get an instance of a service manually or with Injected<T>

There are four common ways to get an instance of a service:

1. **ServiceLocator.Current.GetInstance<T>**: manually retrieve the provider for the service T.

```
IContentLoader loader = ServiceLocator.Current.GetInstance<IContentLoader>();
var page = loader.Get<PageData>(ContentReference.StartPage);
```

2. **Injected<T>**: define a field that will be automatically instantiated.

```
private Injected<IContentLoader> injectedLoader;
public void SomeMethod()
{
  var page = injectedLoader.Service.Get<PageData>(ContentReference.StartPage);
}
```


How to get an instance of a service with constructor dependency injection

3. **SomeController(T param)**: constructor parameter injection.

```
private readonly IContentLoader loader = null;
public MuppetPageController(IContentLoader loader)
{
    this.loader = loader;
}
public void SomeMethod()
{
    var page = loader.Get<PageData>(ContentReference.StartPage);
}
```

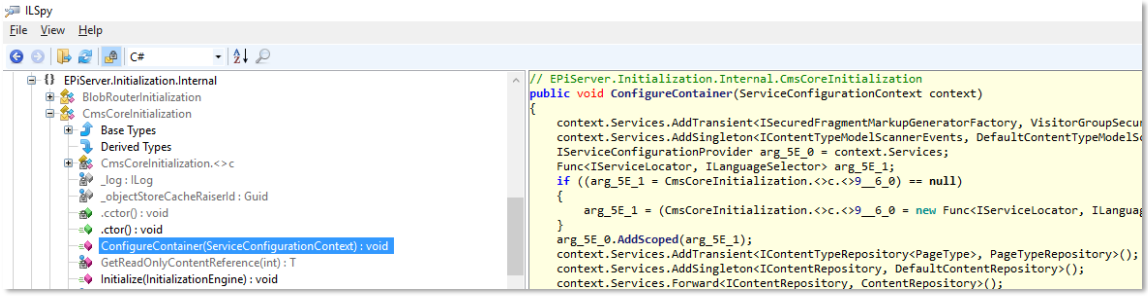
Warning! This third option requires **StructureMap** or some other dependency resolver to be configured. This is the best option for making it easy to unit test and remove dependencies.

How to get an instance of a service in an initialization module

4. Initialization modules do not allow constructor parameter injection, so you could use the initialization engine's **Locate.Advanced** object instead:

```
private IContentLoader loader = null;
private IContentEvents events = null;

public void Initialize(InitializationEngine context)
{
    this.loader = context.Locate.Advanced.GetInstance<IContentLoader>();
    this.events = context.Locate.Advanced.GetInstance<IContentEvents>();
}
```


Dependency injection for default services




```
context.Services.AddTransient
    <IContentTypeRepository<PageType>, PageTypeRepository>();
context.Services.AddSingleton<IContentRepository, DefaultContentRepository>();
context.Services.AddSingleton<IContentLoader, DefaultContentLoader>();
```


Replacing a dependency service

```
using EPiServer.Framework.Localization;
using EPiServer.ServiceLocation;
using System;
using System.Collections.Generic;
using System.Globalization;
```

To replace the default implementation for a dependency service, you have two choices:

1. Apply an attribute to your replacement class:

```
[ServiceConfiguration(typeof(LocalizationService))]
public class MyLocalizationService : LocalizationService
{
```

2. Register your class in an initialization module (see next slide).

Replacing a dependency service

```
public class MyVersionRepository : DefaultContentVersionRepository
{
    public override ContentVersion Load(ContentReference contentLink)
    {
        //add your own logic here
        return base.Load(contentLink);
    }
}
```

Override and add your own implementation

```
[InitializableModule]
public class MyInitialization : IConfigurableModule
    {
        public void ConfigureContainer(ServiceConfigurationContext context)
        {
            context.Container.Configure(ce => ce.For<IContentVersionRepository>().Use<MyVersionRepository>());
        }

        public void Initialize(EPiServer.Framework.Initialization.InitializationEngine context)
        {
            var contentVersionRepository = ServiceLocator.Current.GetInstance<IContentVersionRepository>();
        }

        public void Uninitialize(EPiServer.Framework.Initialization.InitializationEngine context) {}
    }
```

Make the system use your component

Initialization system

- Discovery – implement the `IInitializableModule` interface and decorate with `[InitializableModule]`
- Order of execution – use `[ModuleDependency(typeof(SomeModule))]` to make sure "SomeModule" is initialized before your module.
- Execute the modules
  - Initialize method called once, if no exception thrown
  - If an exception occurs initialization is stopped, retry at next incoming request

Make sure that you fully understand the initialization process. It has a big impact on performance.
http://world.episerver.com/documentation/developer-guides/CMS/initialization/

How to remove the suggested page types feature

IMHO, an annoying feature of Edit view is that when adding a new page, the first group is named Suggested Page Types and contains recently used page types. You can remove this by ejecting the implementation of IContentTypeAdvisor from dependency injection in an initialization module.

```
[InitializableModule]
[ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
public class RemoveSuggestedPageTypesInitializationModule : IConfigurableModule
{
    public void ConfigureContainer(ServiceConfigurationContext context)
    {
        context.Container.EjectAllInstancesOf<IContentTypeAdvisor>();
    }
    public void Initialize(InitializationEngine context) { }
    public void Uninitialize(InitializationEngine context) { }
}
```


Events exposed by the IContentEvents interface

Before events	After events
CreatingContent	CreatedContent
CheckingInContent	CheckedInContent
SavingContent	SavedContent
PublishingContent	PublishedContent
MovingContent	MovedContent
LoadingChildren (GetChildren)	LoadedChildren FailedLoadingChildren
LoadingContent	LoadedContent FailedLoadingContent
LoadingDefaultContent (GetDefaultContent)	LoadedDefaultContent
DeletingContent	DeletedContent
DeletingContentLanguage (DeleteLanguageBranch)	DeletedContentLanguage
And many more…	And many more…

How to listen for content events (1 of 2)

```
[InitializableModule]
[ModuleDependency(typeof(EPiServer.Web.InitializationModule))]
public class PreventPublishingInitializationModule : IInitializableModule
{
    private bool executed = false;
    private IContentEvents events;
    public void Initialize(InitializationEngine context)
    {
        if (!executed)
        {
            events = ServiceLocator.Current.GetInstance<IContentEvents>();
            events.PublishingContent += Events_PublishingContent;
            executed = true;
        }
    }
```

How to listen for content events (2 of 2)

```
    private void Events_PublishingContent(object sender, EPiServer.ContentEventArgs e)
    {
        if ((e.Content as PageData).Name.ToLower().Contains("bad word"))
        {
            e.CancelAction = true;
            e.CancelReason = "Content names cannot contain \"bad word\".";
        }
    }
    public void Uninitialize(InitializationEngine context)
    {
        events.PublishingContent -= Events_PublishingContent;
    }
}
```


Logging

- The Episerver log often reveals issues.
  - When contacting Episerver developer support they probably want to take a look at a log file.
- Logging API shipped with EPiServer is an abstraction for writing log messages
- If you are currently using log4net for logging and want to start using the new API in an existing project, there is a dedicated namespace called `EPiServer.Logging.Compatibility` that will help with the migration.
- Set up located in `EpiserverLog.config`

Logging recommendations

- **Log level**: Error, Debug, Info or All
  - When logging too much it reduces performance, makes the log files large and hard to read. Log only what you need.
- **Store log files on separate drive**
  - You might run out of space
- **Split up into log files each day**
  - One good way to make the files more easy to read
  - log4net.Appender.RollingFileAppender

Enable logging example

```
<log4net>
    <appender name="errorFileLogAppender" type="log4net.Appender.FileAppender" >
      <file value="c:\\EpiserverLog\\1\\Monitor\\Errorlog-file.txt" />
        <encoding value="utf-8" />
        <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
        <appendToFile value="true" />
        <layout type="log4net.Layout.PatternLayout">
            <conversionPattern value="%date [%thread] %level %logger: %message%n" />
        </layout>
    </appender>
    <root>
        <level value="Debug" />
        <appender-ref ref="errorFileLogAppender" />
    </root>
</log4net>
```

Use RollingFileAppender or similar to prevent one large log file

Log file should not be located on same hard drive as the site

Set debug level to Error or Warn to avoid noise

This is a basic example with the minimum needed in a development environment

In a production environment the logging needs to be better configured, for example:

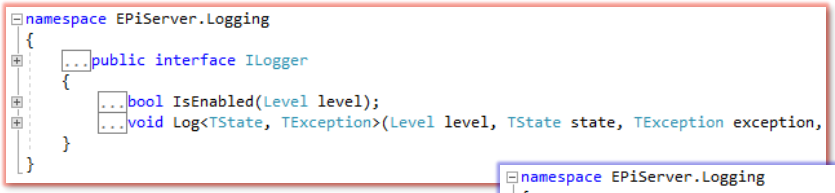
Module A – Reviewing Episerver CMS Fundamentals – Logging

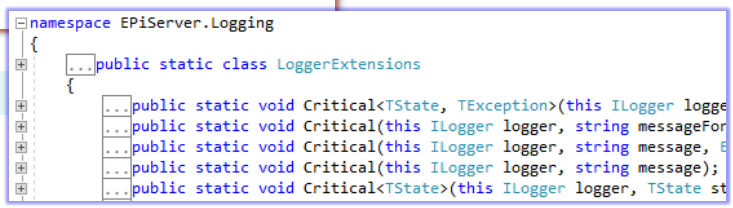
Write to log example

Get the logger service with `LogManager`:

```
using EPiServer.Logging;
private readonly ILogger logger = LogManager.GetLogger();
```

CMS 11 breaking change
It is no longer supported to get an `ILogger` instance from IOC container.





Write to the log:

```
logger.Critical("My message");
```

Episerver

Module A – Reviewing Episerver CMS Fundamentals – Security

Security topics

- Roles and views
- Access rights
- Security in the Episerver project templates
- Security features
- Separate editing server
- Secure the UI folder
- Remove configuration that is not used
- For a full detailed list of security features see the Security section in the Episerver CMS SDK.

Episerver Trust Center
http://www.episerver.com/about/privacy/trust-center/

Episerver 94

Authentication and authorization

Episerver CMS can use either **ASP.NET Membership** (2005) or **ASP.NET Identity** (2013) for authentication and authorization.

To enable ASP.NET Membership aka "Forms"

```
<authentication mode="Forms">
  <forms name=".EPiServerLogin" loginUrl="Util/login.aspx"
         timeout="120" defaultUrl="~/" />
</authentication>
```

To enable ASP.NET Identity

You will also need an OWIN Startup class.

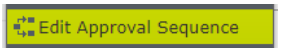
```
<authentication mode="None">
  <forms name=".EPiServerLogin" loginUrl="Util/login.aspx"
         timeout="120" defaultUrl="~/" />
</authentication>
```

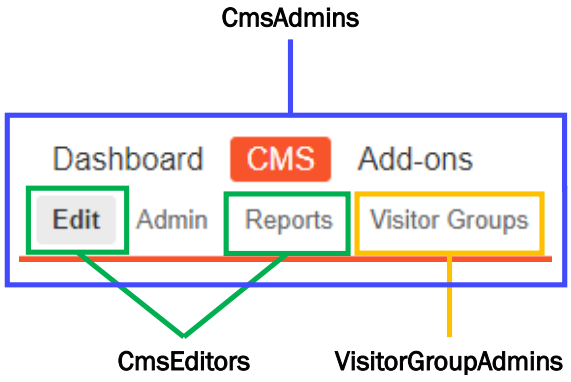

Access to working areas

Every logged in user has access to their own customizable **Dashboard**.

Access to other working areas is controlled by membership of these virtual roles:

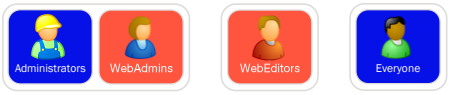
- **CmsEditors**: access to CMS Edit and CMS Reports.
- **VisitorGroupAdmins**: access to CMS Visitor Groups.
- **CmsAdmins**: access to all CMS working areas.
- **EPiBetaUsers**: access to beta features, like **Edit Approval Sequence** menu in CMS 10.1 to 10.8:





The **Add-ons** store and its associated virtual role named **PackagingAdmins** have been removed in Episerver CMS 11. Some screenshots may still show the Add-ons menu,

Stored groups/roles and virtual roles



- **WebAdmins** and **Administrators**: mapped to **CmsAdmins** in Web.config.
- **WebEditors**: mapped to **CmsEditors** in Web.config.
- **Everyone**: Read access to all content.

Good practice: assign rights to a virtual role, not a stored role/group. This provides more flexibility.

```
<episerver.framework>
  <virtualRoles addClaims="true">
    <providers>
      <add name="CmsAdmins" type="EPiServer.Security.MappedRole, EPiServer.Framework"
           roles="WebAdmins, Administrators" mode="Any" />
      <add name="CmsEditors" type="EPiServer.Security.MappedRole, EPiServer.Framework"
           roles="WebEditors" mode="Any" />
      <add name="Everyone"
          type="EPiServer.Security.EveryoneRole, EPiServer.Framework" />
```


Access rights

Episerver user interface uses the term **access rights**. Developers use the enum **AccessLevel**.

What are the six access rights?

	Read	Create	Change	Delete	Publish	Administer
Administrators	☑	☑	☑	☑	☑	☑
Everyone	☑	☐	☐	☐	☐	☐

```
namespace EPiServer.Security
{
    public enum AccessLevel
    {
        NoAccess = 0,
        Read = 1,
        Create = 2,
        Edit = 4,
        Delete = 8,
        Publish = 16,
        Administer = 32,
        FullAccess = 63,
        Undefined = 1073741824
    }
}
```

Does **Administer** access rights give access to the **Admin** view?

- No

So what *does* it do?

- It allows someone *without* access to **Admin** view to set access rights.

Name	Alloy Plan	Visible to	Everyone Manage
Name in URL	alloy-plan Change	Languages	en

Security in the Episerver project templates

The Episerver CMS 11 Empty project template configures the **MultiplexingMembershipProvider** to use **SqlMembershipProvider** as provider1, and **WindowsMembershipProvider** as provider2 because provider1 must support read/write if the Admins need to be able to manage users and roles through the Episerver UI.

The SQL provider is read/write. The Windows provider is read-only.

You can configure additional or alternative providers including *Active Directory* and *ASP.NET Identity*:
http://world.episerver.com/documentation/developer-guides/CMS/security/episerver-aspnetidentity/

Warning! If a user is NOT a member of the virtual role **CmsAdmins** or **CmsEditors**, then login will fail without an explanation.

Security in Alloy (MVC) site

The Episerver CMS Visual Studio Extension project template for the Alloy (MVC) site clears all the membership or role providers, and sets authentication mode to None. In combination with an OWIN startup class (next slide), this activates the **ASP.NET Identity** claims-based authentication system.

```
<authentication mode="None">
  <forms name=".EPiServerLogin" loginUrl="Util/login.aspx"
         timeout="120" defaultUrl="~/" />
</authentication>
```

```
<membership><providers><clear /></providers></membership>
<roleManager><providers><clear /></providers></roleManager>
```

It also creates the **WebAdmins** group for you, gives the group full rights, and the first time you browse to the new website on the local server machine, it prompts you to create an **Admin** user.

	Name	Provider
	WebAdmins	EPi_AspNetIdentityRoleProvider

	Name	Provider
	Admin	EPi_AspNetIdentityUserProvider

OWIN Startup in Alloy (MVC) site

```
using EPiServer.Cms.UI.AspNetIdentity;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
using System;
using System.Web;
```

```
[assembly: OwinStartup(typeof(Cms10Site.Startup))]
public class Startup
{
    public void Configuration(IAppBuilder app)
    {   // Add CMS integration for ASP.NET Identity
        app.AddCmsAspNetIdentity<ApplicationUser>();
        // prompt to register an admin account if browser is local on server
        app.UseAdministratorRegistrationPage(()
            => HttpContext.Current.Request.IsLocal);
        app.UseCookieAuthentication(new CookieAuthenticationOptions
        {
            AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
            LoginPath = new PathString(Global.LoginPath), // and so on
```


Security features

- **Authentication and authorization**: It is easy to create your own provider for any type of user database. Note that where and how user credentials are stored, depends entirely on the authentication model used. http://world.episerver.com/documentation/developer-guides/CMS/security/Authentication-and-authorization/
- **Injection**: All code in Episerver CMS use parameterized API's to make sure that injection attacks cannot be carried out from untrusted input. There are no code paths in Episerver CMS that uses untrusted data in XML-related calls.
- **Cross-site scripting**: The editorial and administrative interfaces are areas where HTML and scripts are sometimes allowed to be posted and used as-is on a web page. Here, Episerver CMS relies on its authorization features to ensure that only trusted users can provide content.

Security features – attack prevention

- **CSRF**: Episerver CMS has a CSRF prevention mechanism that automatically detects forged requests for all system pages. The event validation mechanism in ASP.NET is also enabled for these pages. If you're using ASP.NET MVC, use the Html.AntiForgeryToken() helper in conjuction with the [ValidateAntiForgeryToken] attribute on your post method.
- **Virus protection**: Episerver CMS relies on third-party products for virus protection. Note that files that are uploaded to the media manager in Episerver CMS, will never be executed by the Episerver CMS system, preventing potential viruses inside files to spread from there to the CMS system.

Common security mistakes

- **Security misconfiguration**: Strong security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults.
- **Insufficient transport layer protection**: Episerver CMS fully supports the use of SSL (HTTPS protocol) for any web page that is associated with a forms-based logon screen, and the use of SSL is strongly recommended.
- **Failure to restrict URL access**: Information presented on public-facing web pages are subject to authorization based on the content that is displayed. In no case does Episerver CMS rely on security through a secret actual URL. It is very important to restrict URL access to, for example, custom Web Services.

Module A – Reviewing Episerver CMS Fundamentals – Security

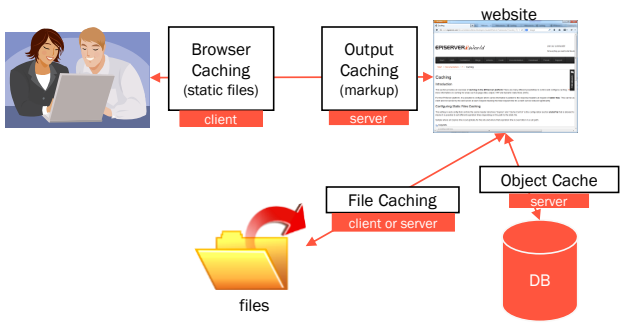
Configure and secure the UI folder path

- If creating a new project from Visual Studio the UI path will be set to **/EPiServer/** by default.
  - Change the UI folder path in web.config.
  - Note that changing the UI path will not in itself make your website or access to the UI more secure.
- There can be more paths to replace if you have other Episerver products such as Commerce installed
- Use **SSL (Secure Sockets Layer)** to secure the website and/or UI folder

http://world.episerver.com/documentation/developer-guides/CMS/security/Securing-edit-and-admin-user-interfaces/

Module A – Reviewing Episerver CMS Fundamentals – Caching

How Episerver caches content



http://world.episerver.com/documentation/developer-guides/CMS/caching/

What are the types of caching?

- **Object (aka page) caching**: when a content item is requested it is loaded from the database and stored in the object cache on the web site server for a minimum of 12 hours (by default).

```
<episerver>
  <applicationSettings pageCacheSlidingExpiration="12:00:00"
```

  The object cache uses in-proc memory by default, but it can be configured to use other providers, for example Redis, for highly performant distributed caching.
- **Output caching**: when an HTTP response is returned from the server, it can be cached. To enable it, (1) apply `[ContentOutputCache]` and configure `<applicationSettings>` in Web.config, or (2) write code to control the Response.Cache object.
- **CDN and Browser caching**: CDNs and browsers look at the HTTP response headers to determine what and how long to cache. Control this through output caching.

Object cache

- Based on Microsoft's ASP.NET Cache.
- Automatically caches all objects in Episerver CMS that are requested via the API, for example using `IContentLoader` or `IContentRepository`.
- Only read-only objects are stored to enable great performance.
- Has a few advanced characteristics to improve scalability. For example, it uses an optimistic locking approach when multiple threads are reading the same data they will all piggyback on the same database calls to avoid putting to much load on the database for "hot" objects that have not yet been cached.
- Use the `ISynchronizedObjectInstanceCache` registered service to manage objects in the cache with custom dependency and eviction policies and to synchronize cache removal among servers in a load-balanced environment using the event system.:
  https://world.episerver.com/documentation/developer-guides/CMS/caching/Object-caching/

Object cache with Remote Events

Remote Events is the Episerver feature that invalidates content items stored in the object cache in a load balanced deployment.

- Cache is emptied when content is published.
- To enable Remote Events, add the following to the episerver element in Web.config:

```
<episerver>
  <sites>
    <site>
      <siteSettings enableEvents="true" enableRemoteEvents="true" />
```

Remote Events can be implemented using:

- WCF via UDP or TCP in an on-premise deployment (see example configuration on next slide)
- Service Bus in an Azure cloud deployment

Configuring Remote Events

```
<bindings>
  <netTcpBinding>
    <binding name="RemoteEventsBinding"
             portSharingEnabled="false">
      <security mode="None"/>
    </binding>
  </netTcpBinding>
</bindings>
```

```
<client>
  <endpoint name="customer-10.11.12.14"
            address="net.tcp://10.11.12.14:5000/RemoteEventService"
            binding="netTcpBinding" bindingConfiguration="RemoteEventsBinding"
            contract="EPiServer.Events.ServiceModel.IEventReplication"/>
</client>
```

Connect to server with IP address: 10.11.12.14

```
<services>
  <service name="EPiServer.Events.Remote.EventReplication">
    <endpoint name="RemoteEventServiceEndPoint"
              contract="EPiServer.Events.ServiceModel.IEventReplication"
              binding="netTcpBinding" bindingConfiguration="RemoteEventsBinding"
              address="net.tcp://localhost:5000/RemoteEventService" />
  </service>
</services>
```

On server with IP address: 10.11.12.13

File caching

- Files delivered from Episerver CMS can be either client cached or server and client cached.
- The server cache uses the kernel cache available in IIS, which makes often requested files to be delivered directly by the IIS kernel which offer superior performance.
- The client cache let's you for instance set a specific folder to be cached for a certain time period.

Caching static application files and cache busting

For performance reasons, it's a good idea to cache static content for about a year.

Use the following in Web.config's `<system.webServer>` element:

```
<staticContent>
  <clientCache cacheControlMode="UseMaxAge" cacheControlMaxAge="365.00:00:00" />
```

But what happens when you want to change the contents of a static file that does not include a version number or date in its name, like **site.css**? We need to "bust" the cached version.

You can write some code that adds a "fingerprint" to each file automatically. This blog article shows and example of how:

http://madskristensen.net/post/cache-busting-in-aspnet

Output caching

- Based on ASP.NET Output Caching.
- This is an effective method since the entire rendered markup of a web page or user control will be cached for a specified duration.
- The cache is automatically invalidated when content in Episerver CMS is published.
- You can define dependency rules for the cache, as well as which parts of the website should be affected.
- Rules for browser (client) caching can be easily configured in Episerver CMS. For instance, you can set all static files delivered by the Episerver CMS to be cached by the client for a certain time period. It is also possible to define how long dynamically generated pages should be cached on the client.

Common questions on output caching

In the CMS Developer Guide, we recommend that output caching be turned off in most cases.

On a site where most of the content is not personalized, no customer logs in, and content is not frequently updated, the content output cache should be on. Otherwise it should be turned off.

Why?

- ContentOutputCache will only work for users who are not logged in.
- ContentOutputCache will be invalidated as soon as any editor change the content.

Module A – Reviewing Episerver CMS Fundamentals – Caching

Full and partial output caching

To enable ASP.NET MVC output caching use Episerver's content-aware **ContentOutputCache** attribute:

```
// response cached for 20 minutes
[ContentOutputCache(Duration = 1200)]
public ActionResult Index(ProductPage currentPage)
```

```
// response cached for 2 hours
[ContentOutputCache]
public ActionResult Index(ProductPage currentPage)
```

```
<episerver>
  <applicationSettings
    httpCacheability="Public"
    httpCacheExpiration="02:00:00"
```

The default httpCacheExpiration is 00:00:00.

The policy only applies to HTTP GET requests from anonymous visitors. Logged in users will never receive cached responses.

Episerver 116

Module A – Reviewing Episerver CMS Fundamentals – Caching

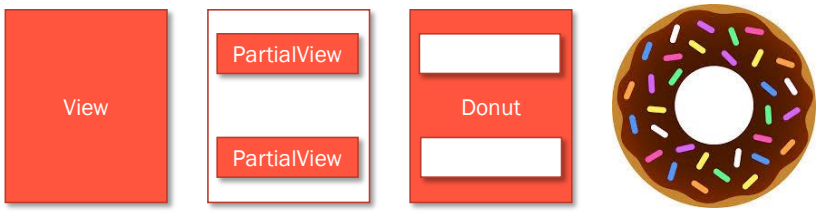
What types of output caching are supported in Episerver?

ASP.NET MVC has built-in support for:

- **View** caching: the entire HTML page is cached.
- **Partial View** caching aka "donut hole" caching: one or more parts of an HTML page are cached.

ASP.NET MVC does NOT have built-in support for:

- "**Donut**" caching: most of the HTML page is cached EXCEPT one or more parts.



Episerver 117

DevTrends' MvcDonutCaching NuGet package

Donut caching is a server-side caching technique in which the entire page is cached, except for small portions that remain dynamic.

ASP.NET MVC Extensible Donut Caching brings donut caching to ASP.NET MVC 3 and later. The code allows you to cache all of your page apart from one or more Html.Actions which can be executed every request. Perfect for user specific content such as personalization by Visitor Groups.

This post describes MvcDonutCaching, a new open-source NuGet package that adds donut caching to MVC3 in a simple and performant manner:
http://www.devtrends.co.uk/blog/donut-output-caching-in-asp.net-mvc-3

Jon Jones has written one especially for Episerver: http://jondjones.com/learn-episerver-cms/episerver-developers-guide/episerver-caching/how-to-implment-a-donut-hole-cache-in-episerver

Making the most of a CDN

For **static** content you should implement a "never expires" policy by setting a far future Expires header. Include a version identifier in the path to the resources, to allow intermediary proxies like CDNs to store and serve them indefinitely, e.g. jquery-3.1.0.min.js

```
Expires: Thu, 15 Apr 2090 20:00:00 GMT
```

For **dynamic** content you should set **cache-control** in the HTTP header, for example,

```
Cache-Control: public, max-age=1200, must-revalidate
```

public means intermediaries should cache the content, **private** would mean only the browser should. **max-age** is an integer value of seconds, so 1200 would mean 20 minutes.

Module A – Reviewing Episerver CMS Fundamentals – Caching

Using Response.Cache to control caching

Set cacheability in the HTTP response headers just before returning an action result:

```
Response.Cache.SetCacheability(HttpCacheability.Public);
Response.Cache.SetExpires(DateTime.Parse("6:00:00PM"));
```

To set a sliding expiration so each response renews the cache:

```
// expire in one minute, with sliding expiration
Response.Cache.SetExpires(DateTime.Now.AddMinutes(1.0));
Response.Cache.SetSlidingExpiration(true);
```

To set a max-age (TimeSpan will be automatically converted into seconds):

```
Response.Cache.SetMaxAge(TimeSpan.FromMinutes(30));
```


Module A – Reviewing Episerver CMS Fundamentals – Good practice

Development good practice

- Keep it simple
- Utilize Episerver's built in functionality
- Follow OWASP top ten list for security, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- Avoid undocumented functionality
- Use correct functionality for the correct purpose
- Develop with fall-backs, code defensively
- Implement logging

"Always, always use Episerver's API when accessing Episerver data. Not only does it protect against security vulnerabilities like SQL-injection, it also adds smart caching, event notifications and other goodies." Frederik Vig, http://www.frederikvig.com/2011/05/part-1-injection-owasp-top-10-for-episerver-developers-3/

Development good practice

Language handling

- When creating a content type, leave the display name blank instead of setting an English default name in the definition. Start using a language resource file for these straight away.
- The same rule goes for the properties as well.

Other tips and tricks

- Activate trace in Web.config to troubleshoot: <trace enabled="true" pageOutput="true" … />
- Learn to use the debugger – it provides a lot of useful runtime information

Performance pointers

- Always use release-compiled code. In MVC, release-compiled code gives 50% better performance
- Minify and bundle JavaScript and CSS files
- Use vector graphics or sprites for images
- One application pool per application is optimal from an access point of view, but one application pool for all applications is better from a performance point of view
- Remember that having more than one version of Episerver on a web server means that more memory is needed

Module A – Reviewing Episerver CMS Fundamentals – Good practice

More good practice for performance and security

Performance optimization
http://vimvq1987.com/2017/06/episerver-cms-performance-optimization-part-1/

Performance tips
https://talk.alfnilsson.se/2015/11/25/code-best-practices-and-performance-optimization-the-summary/

Security checklist for EPiServer website
http://world.episerver.com/blogs/Daniel-Ovaska/Dates/2015/6/security-checklist-for-episerver-website/

Recommendations for ASP.NET security settings
http://world.episerver.com/documentation/developer-guides/CMS/security/Recommendations-for-ASPNET-security-settings/

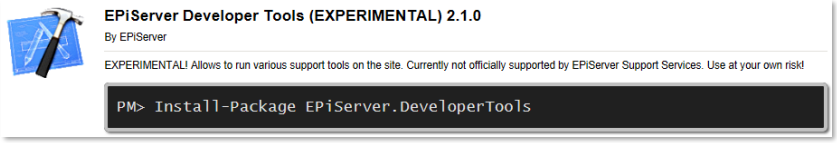
Common security issues with a CMS application
http://world.episerver.com/blogs/Petra-Liljecrantz/Dates/2016/4/common-security-issues-with-a-cms-application/

Module A – Reviewing Episerver CMS Fundamentals – Good practice

Episerver developer tools

- View contents of the StructureMap container
- View Log4net logs (based on in-memory appender)
- View Content Type sync state between Code and DB
- Take memory dumps, and view loaded assemblies in AppDomain
- View all registered view engines, templates for content, and view and test ASP.NET routes
- View startup time for initialization modules
- View remote event statistics, provider and servers

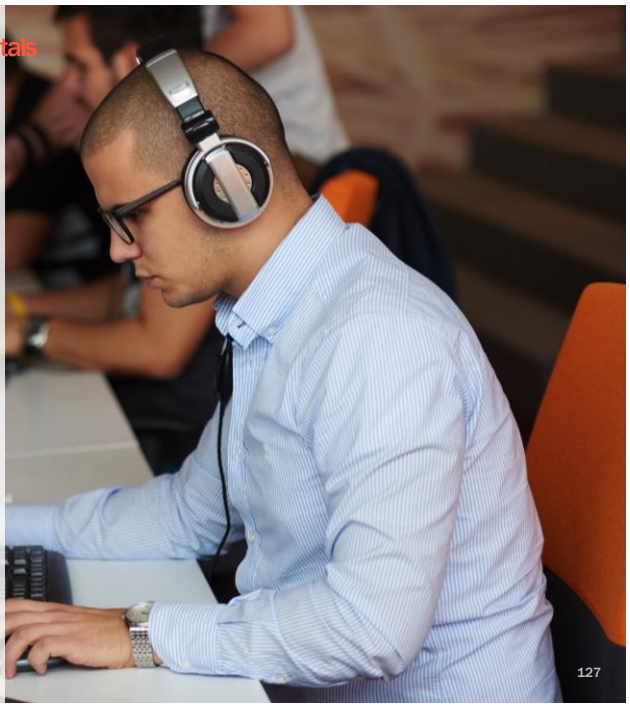
https://github.com/episerver/DeveloperTools



Module A – Reviewing Episerver CMS Fundamentals

Exercises for Module A

1. Validating content and properties.
2. Exploring authentication and authorization.
3. Completing some challenges:
   a) Create and use some blocks
   b) Initialization, event handling, and complex validation
   c) Site-level page property
4. Exploring Episerver developer tools.



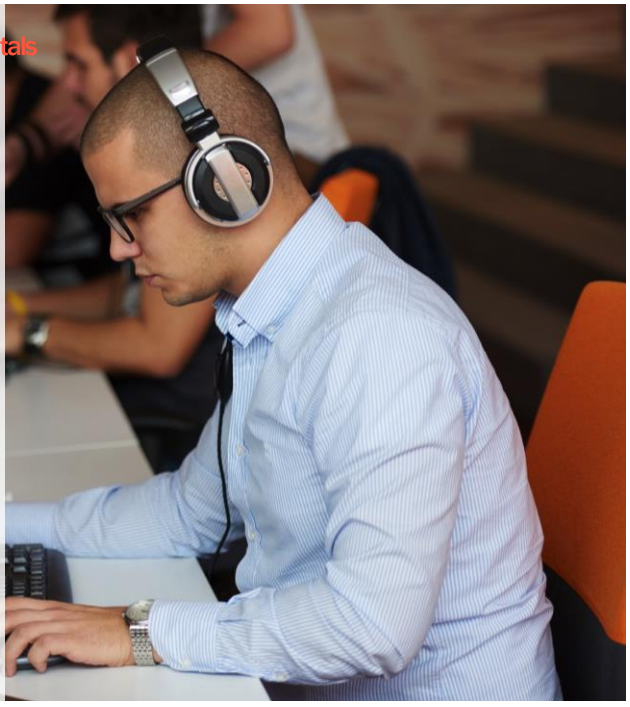
Episerver

127

Module A – Reviewing Episerver CMS Fundamentals

Optional: Exercises for Practice

1. Create an empty Episerver site and follow recommended practice
2. Create some page types with properties
3. Create some block types
4. Create content templates for pages and blocks
5. Create pages and blocks
6. Extend the site
7. Optimize the site



Episerver

Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

1. Create an empty Episerver site and follow recommended practice

1. Create an Empty Episerver web site with CMS Search capability.
2. Update to the latest CMS version.

In the next few slides you will create content types. Follow recommended practice by:

- ensuring all content types and properties are grouped and ordered appropriately,
- set group and tab names using site-defined tab and group definitions that include: “SEO”, “Site Settings”, “Event Info”, “Practice Site”,
- use XML language file localization for all display names and descriptions of all your content types properties, groups, and tabs, localized into English and Swedish, and
- add an image (120x90) shown when editors create all page and block types.

Episerver

Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

Translations

English	Swedish
Event Info	Händelseinfo
Site Settings	Webbplatsinställningar
Practice Site	Praxis Webbplats
Start	Börja
This is the home page for the site. It has settings that apply to the shared site layout.	Detta är hemsidan för webbplatsen. Det har inställningar som gäller för den delade webbplatslayout.
Standard	Grundläggande

English	Swedish
The standard page has an introductory paragraph of text and a main body.	Standard sida har en inledande stycket i text och en huvuddel.
Main introduction	Huvud introduktion
Brand logo image	Logotyp bild
Main body	Huvudkroppen

Episerver

Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

2. Create some page types with properties

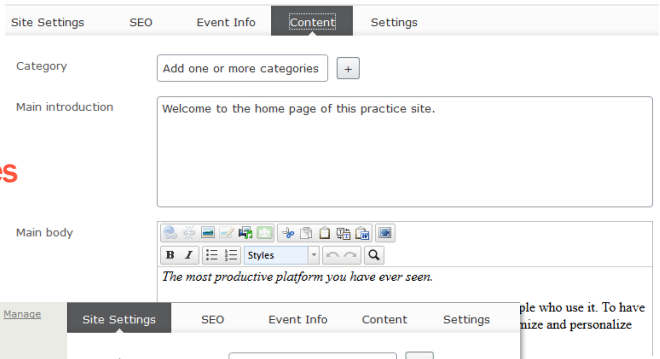
Page	Properties	Restrictions
SitePageData*	MetaTitle	5 to 60 characters
	MetaDescription	Required
	MainIntro	Plain text
	MainBody	Rich text
StartPage	BestBet	Points to a single page of any type except a StartPage.
	FavoritePagesMenu	Points to multiple pages of any type except a StartPage.
	Logo	Points to any image media asset.
	ContactEmail	The e-mail address of someone to contact about the site.
StandardPage	BlocksAndPages	Any combination of blocks and page (partials).

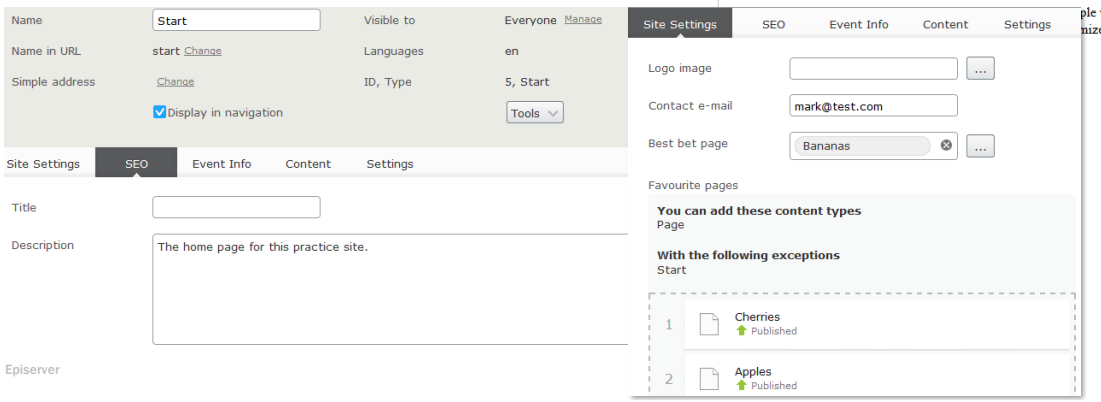
Episerver *An abstract base class that the other two page types derive from.

Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

2. Create some page types with properties

The StartPage's All Properties view should look something like this:

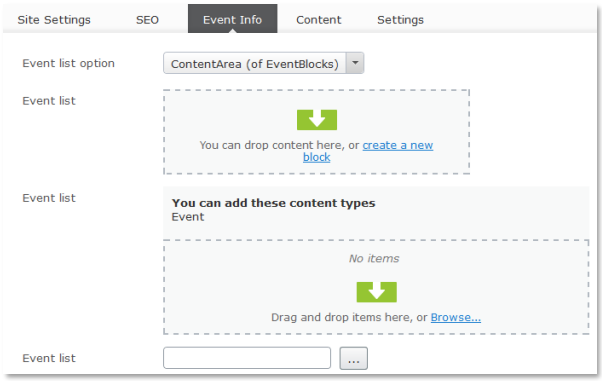




Episerver

3. Create some block types

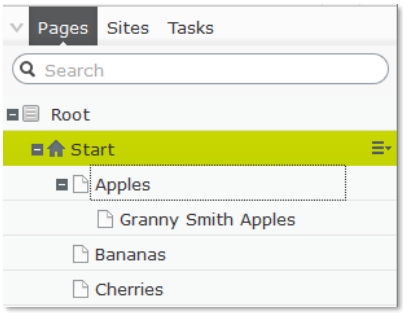
1. Create a block type named **EventBlock** with properties: **Title**, **When**, **Where**, **Description**.
2. Create a block type named **PersonBlock** with: **FirstName**, **LastName**, **BirthDate**, **Summary**.
3. Add a property to the **StartPage** named **EventList** that can contain any number of instances of the **EventBlock**. There are several ways to implement this:
   a) A content area,
   b) An IList of content references, or
   c) A reference to an asset folder that will contain some blocks.



4. Create content templates for pages and blocks

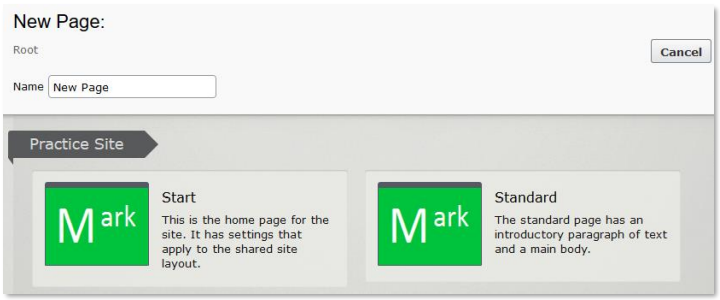
1. Create a view model interface and a default implementation for it. The view model should have a property called **CurrentPage** that can be any type that derives from SitePageData, and a property named **HomePage** that returns the *content* (not a reference) for the web site's start page.
2. Create a shared layout that outputs the **Logo**, hyperlinks to **BestBet**, pages in the **FavouritePagesMenu**, and contact e-mail in the footer. The layout should output the META values in the head element along with a canonical link for the page.
3. Create suitable page templates for the page types (remember to use the view model):
   a) In the page template for both the start and standard pages, if **MainIntro** is empty, output **MetaDescription** instead.
   b) In the page template for the start page, provide visitors with the ability to control the sort order that the events are shown: by title, when, or where.

Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice



5. Create pages and blocks

1. Run the site, create an instance of the Start page and set it as the start page for the site.
2. Create at least three instances of the standard page underneath the start page and at least two other standard pages underneath of existing standard pages to build a hierarchy.
3. In the start page's site settings, select one standard page for the best bet. Select multiple pages for the favourite pages menu, upload an image and select it for the logo, and enter an e-mail address.



Episerver

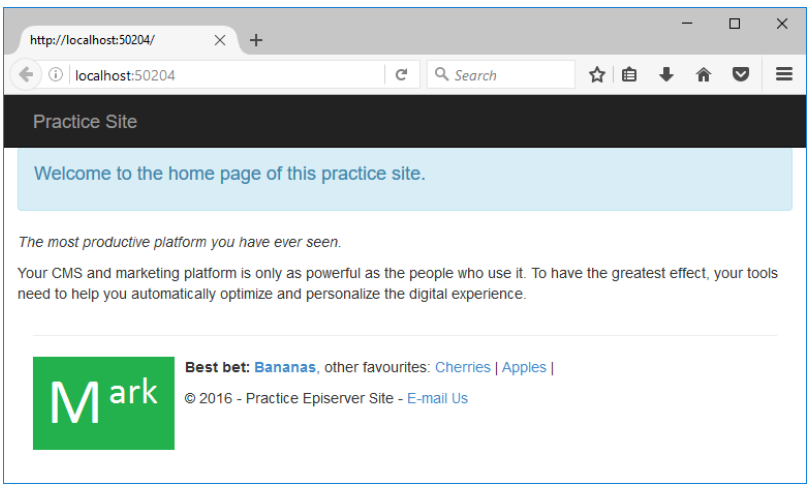
Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

5. Create pages and blocks

Your site should look something like this:

Is the META description used as a fall back for the introductory text in the body of the page?

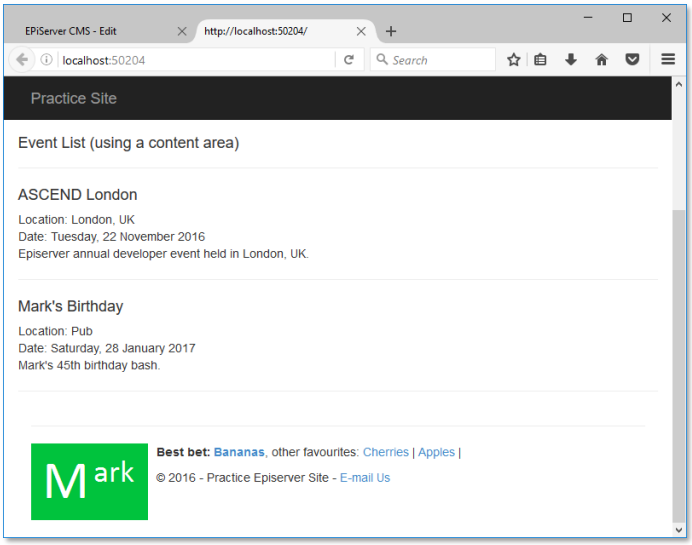
Can it be replaced with different content?



Episerver

5. Create pages and blocks

1. Create at least three instances of the event block and add them to the start page.
2. Create an instance of the person block and test that it CANNOT be used in the event list.



Module A – Reviewing Episerver CMS Fundamentals – Optional: Exercises for Practice

6. Extend the site

1. Create a validation mechanism that prevents setting a BestBet if the page is already in the list of FavouritePagesMenu, or vice versa, and then test it.
2. Create a Scheduled Job that finds pages that contain “iWatch” in the MetaDescription or MainIntro. If found, change to “Apple Watch” and log the fact the change was made.
3. Create an Initialization Module that listens for when content is published and prevents non-administrators from using the phrase “iPhone 8” until after 1st September 2017.
4. Allow the visitor to search for content that they have access to.
5. On the Start page view, output the following for the BestBet page: the names of its children, its ancestors, and its descendants. The dates and statuses of its versions.
6. Create a CodeOnly page type that is not available in Edit view. On the Start page allow anyone to populate a form and submit it to create a CodeOnly page under the BestBet.

7. Optimize the site

Modify the start and standard page views to show a warning message if the MainIntro is being replaced by the META description but only when editing.

Add a new property to SitePageData named PrefetchLinks that contains one or more links that should be pre-fetched by inserting <link> elements into the <head> like this:

```
<link rel="prefetch" href="/en/about-us">
```


Module B
Working with Content using APIs

Content generation often needs to be automated to, for example, minimize the work for the editor or to allow for user-submitted content. To handle this you need know how to work with the content programmatically.

Module B – Working with Content using APIs

Topics

- Overview
- Managing content with APIs
- Getting, Filtering, & Finding
- Sharing content
- Programming Content Approvals
- Programming Notifications
- Programming Key Performance Indicators
- Other APIs

Module B – Working with Content using APIs – Overview

What is content?

CMS

- Pages: a class that derives directly or indirectly from `PageData`.
- Blocks: a class that derives directly or indirectly from `BlockData`.
- Media assets: a class that derives directly or indirectly from `MediaData` (`ImageData`, `VideoData`).

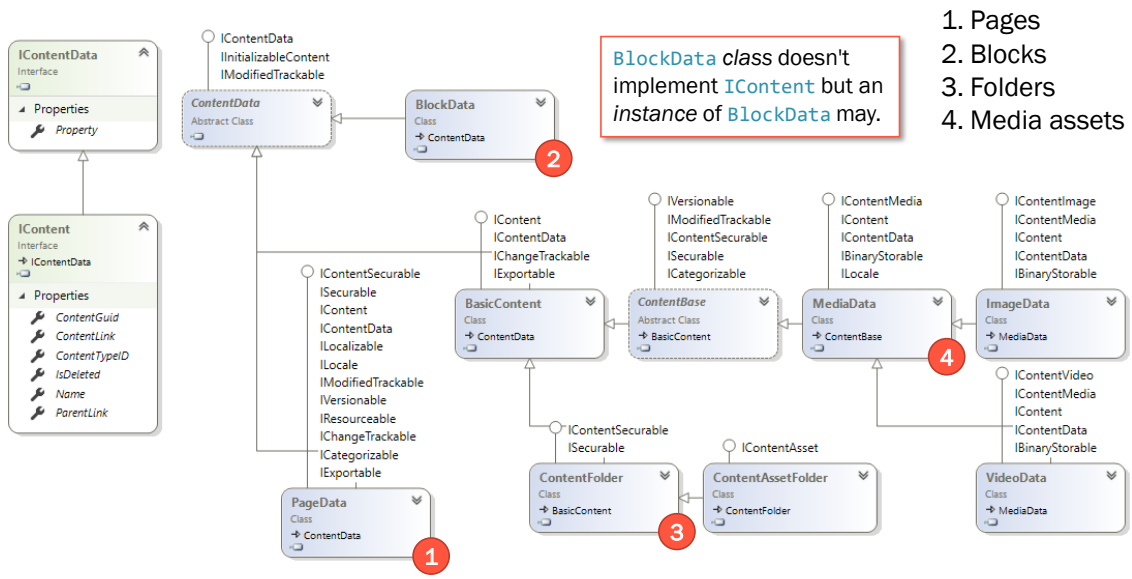
Commerce

- Catalogs
- Products
- Variants

All content must be decorated with [`ContentType`] and ”implements” `IContent`

Important content-related interfaces

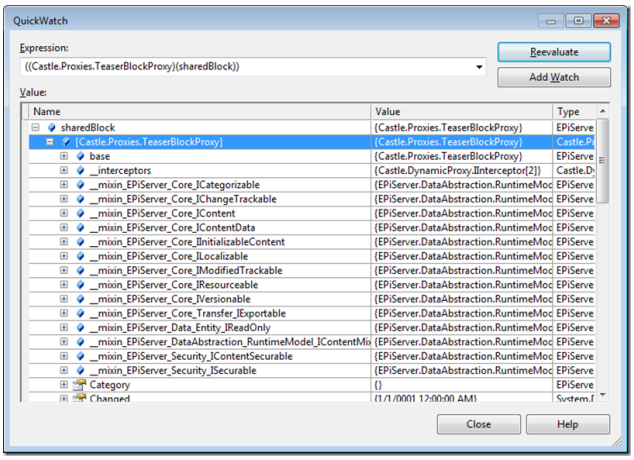
Interface	Description
`IContentData`	An object (not necessarily content) that has a dictionary of properties.
`IContent`	A named instance of a type of content that can be referenced and has a parent.
`ILocalizable`	Has language branches like English and Swedish.
`IVersionable`	Has start and stop publish dates and a state.
`ISecurable`	Has access rights.
`IBinaryStorable`	Has an associated BLOB stored outside the database.
`IInitializable Content`	Has a SetDefaultValues method that can initialize properties to values when created.
`IChangeTrackable`	Has properties: Changed, ChangedBy, Deleted, DeletedBy, Saved, and so on.



 http://world.episerver.com/blogs/Johan-Bjornfot/Dates1/2012/11/Shared-blocks--IContent/

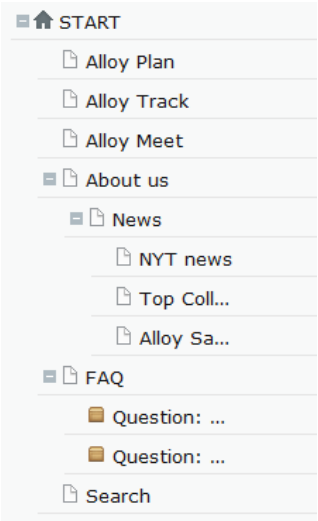
Castle proxies used so shared blocks "implement" IContent

If you attach a debugger and look at the instance returned from `GetDefault<T>` (where T is a type inheriting `BlockData`) you can see that the `TeaserBlockProxy` instance implements `IContent`, even though the `TeaserBlock` class does not.



Handling the content structure

- Why?
  - Easy to work with
  - No performance problems
- How?
  - Structure depth
  - Amount in each node
- When?
  - Always
  - Especially when items are programmatically created



Controlling the page tree depth and children

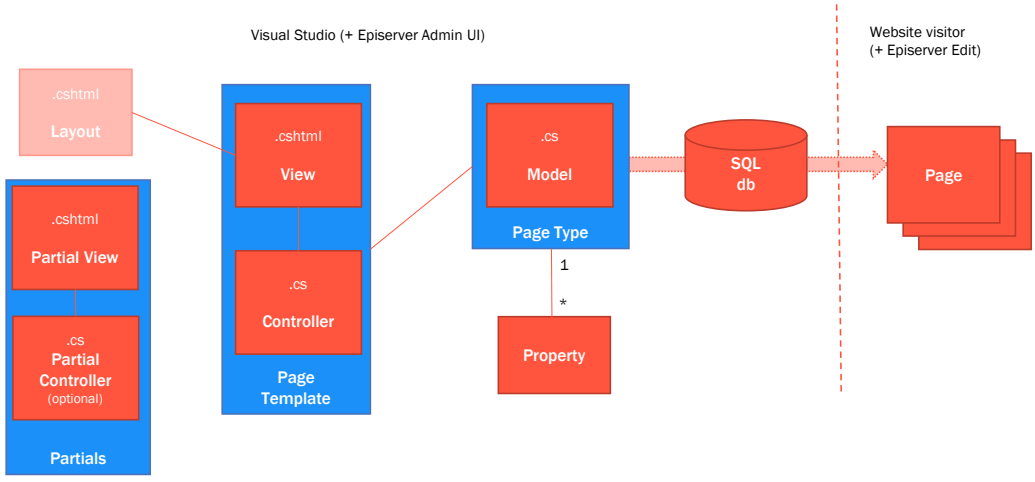
If content editors complain about the page tree being slow, explore using **PowerSlice** add-on to treat all content as if stored in an indexed "bucket".

Recommendation: less than 100 children in each page node.

Especially important when content is created programmatically based on user actions, for example:

- The editor is creating news pages from the UI.
- The developer has added code for a "create news page" button and logic that will automatically place the page in the corresponding folder in the page tree: `/About Us/News/`
- This saves work for the editor as they do not need to find the correct place in the tree before they add the page, but it can also result in a very large amount of pages under the same node, which in its turn could make it more difficult to find what you are looking for and in extreme cases also cause performance problems. You should consider adding additional levels: `/About Us/News/<year>/<month>/<day>`

How it all comes together



How it all comes together – page types and pages

Page type

- A page type defines a set of properties.
- Through these properties the page type defines the type of content and the way in which content can be entered into a page.
- Page types can be created from code or from the administration interface.

Page

- A page is an instance of the .NET class that defined the page type.
- Pages are used by editors in edit view to create the actual pages and fill them with content.
- When creating a page the editor assigns values to the properties defined by the page's page type.

How it all comes together – strongly-typed models

The page system in Episerver CMS supports strongly typed models.

This means that when a page is requested, the instance will be created as the model type that is associated with the PageType in question. The APIs contain generic classes and methods to return typed PageData objects and there is also the possibility of defining PageTypes through annotations in code.

The detection of code-defined PageTypes is handled via class and property attributes. During site initialization all assemblies in the bin folder are scanned and all class types derived from EPiServer.Core.PageData are passed to the synchronization engine.

The annotation information is constructed by merging the annotated settings with the settings stored in the database using the administrative interface. Any automatic properties on your typed page class will reflect the values of the backing PropertyData collection without the need of writing any code.

Module B – Working with Content using APIs – Overview

How it all comes together – page templates

When a page is requested by a visitor, the most suitable page template that matches the context and is associated with the page's page type is used for displaying the content of the page.

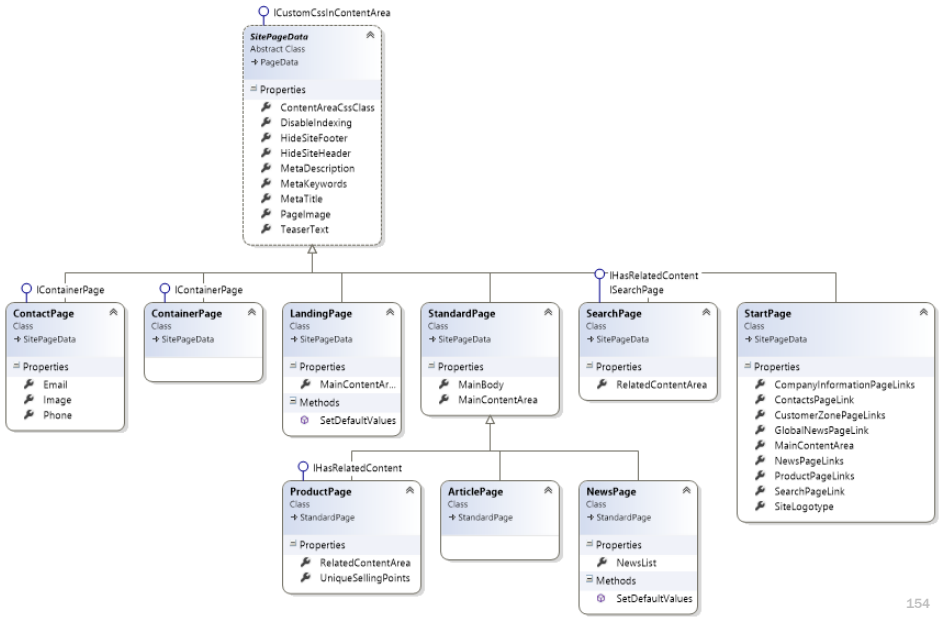
A page type can be associated with multiple page templates, which is useful when publishing content in multiple display channels.

Page templates usually consist of markup, server-executed HTML helper methods, and static text.

Module B – Working with Content using APIs – Managing content with APIs

Alloy

Add a **Class Diagram** to your project to document your page type inheritance hierarchy, for example, like this one for the Alloy web site:



IContent concepts

- The most important for a developer to grasp about IContent and IContent instances:
  - It can be coupled with a template and **Rendered** to visitors
  - It is **Editable**
  - It is **Stored** in the **database**

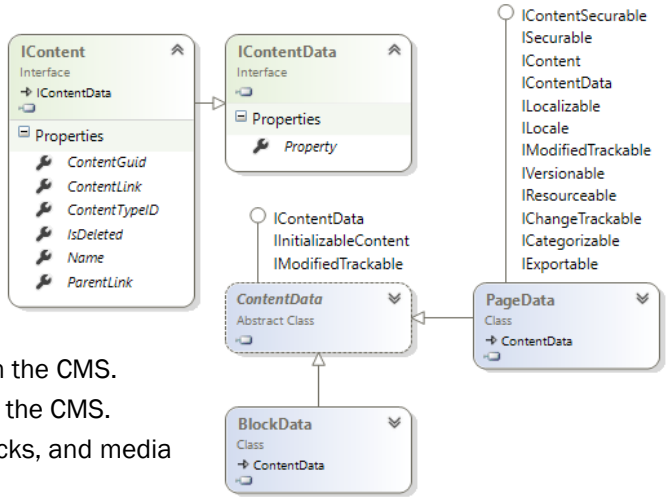
- When you create a shared block programmatically: why does it need to be cast to IContent before it can be saved?
  - Blocks can be used as property types as well as shared. Only shared blocks need to implement IContent.

Content model

In older versions of Epierver CMS, all content was an instance of PageData.

Since CMS 7, the content model is more flexible.

- **IContent**: represents identifiable content in the CMS.
- **IContentData**: represents the properties in the CMS.
- **ContentData**: abstract class for pages, blocks, and media such as images and video.
- **PageData**: represents a page.
- **BlockData**: represents block content.
- **MediaData**: represents a media asset.



PageData and MediaData implement IContent.
ContentData and BlockData do NOT.

Programmatically creating and saving a new page

- Generate a new content item using `IContentRepository`, setting its parent:

```
IContentRepository repo =
    ServiceLocator.Current.GetInstance<IContentRepository>();
NewsPage newsPage = repo.GetDefault<NewsPage>(parentLink: PageReference.StartPage);
```

- Set required page properties:

```
newsPage.Name = "Today's news";
newsPage.MainBody = new XhtmlString("<p>This is produced
```

- Save the page with appropriate save action and access level:

```
ContentReference newPagesRef = repo.Save(newsPage,
    EPiServer.DataAccess.SaveAction.Publish,
    EPiServer.Security.AccessLevel.NoAccess);
```

```
namespace EPiServer.Security
{
    public enum AccessLevel
    {
        NoAccess = 0,
        Read = 1,
        Create = 2,
        Edit = 4,
        Delete = 8,
        Publish = 16,
        Administer = 32,
        FullAccess = 63,
        Undefined = 1073741824
    }
}
```


AccessLevel enum

When calling IContentRepository.Save method, you can pass an AccessLevel. This is the minimal level that the current user (i.e. anonymous or logged in visitor or editor) must have in order for the method to succeed. If the current user does not have that minimal level then the Save method throws an exception. Therefore, if you pass the NoAccess value, you are allowing every user to successfully call the method.

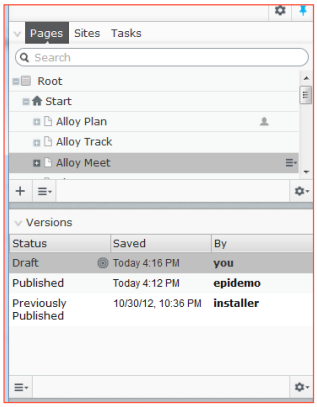
Content versions and states

A content item that supports different statuses implements **IVersionable**.

The interface contains a property **Status** that specifies the current status of the content version. A content version can have one of the following different statues:

- **NotCreated**
- **CheckedIn**, **CheckedOut**
- **AwaitingApproval**, **Rejected**
- **DelayedPublished**, **Published**, **PreviouslyPublished**

A state transition can programmatically be specified by parameter **SaveAction** in the call to **IContentRepository.Save** (see next slide).



CMS 10 and later Save API improvements

One of the improvements in CMS 10 is to the **IContentRepository** Save API. The **SaveAction** enum has an option of **Save**. This has now been deprecated (it won't appear in IntelliSense but it would compile).

- **CheckIn** - Checks in a version indicating that it is ready to be published
- **CheckOut** - Checks out a version to indicate that it is being worked on. (New in CMS 10)
- **RequestApproval** – Indicate that the version is ready for an approval review.
- **Reject** - Rejects a version. This is normally done after a review has been done.
- **Schedule** – Used to schedule a version for automatic publishing at a later date. (New in CMS 10)
- **Publish** - Publishes a version. The currently published version will automatically transition to a previously published state.

Improving the Save experience in CMS 10

http://world.episerver.com/blogs/Henrik-Nystrom/Dates/2016/10/improving-the-saving-experience/

Programmatically updating an existing page

Content is read-only when retrieved through **Get** method(s) so:

1. Call **CreateWritableClone** method of any **IReadOnly** object to be able to make changes.
2. Set required page properties.
3. Save the page with an appropriate save action and access level depending on scenario.

```
ContentReference pageLink = ...;
var repo = ServiceLocator.Current.GetInstance<IContentRepository>();
NewsPage newsPage = repo.Get<PageData>(pageLink).CreateWritebleClone() as NewsPage;
if (newsPage != null)
{
    newsPage.MainBody = new XhtmlString("<p>This was updated programmatically.</p>");
    repo.Save(newsPage, EPiServer.DataAccess.SaveAction.CheckIn,
        EPiServer.Security.AccessLevel.Edit);
```


Programmatically creating new shared blocks

Creating a shared block is similar to creating a page, except BlockData-derived classes do not implement **IContent** so you must cast the block instance to **IContent** before you can set the **Name** property or call the **Save** method:

```
ContentReference forAllSites = ContentReference.GlobalBlockFolder;
var repo = ServiceLocator.Current.GetInstance<IContentRepository>();
var editorial = repo.GetDefault<EditorialBlock>(parentLink: forAllSites);
editorial.MainBody = new XhtmlString("<p>Hello World!</p>");
var content = editorial as IContent;
content.Name = "MyNewSharedBlock";
ContentReference newBlocksRef = repo.Save(content,
    EPiServer.DataAccess.SaveAction.Publish,
    EPiServer.Security.AccessLevel.NoAccess);
```


Programmatically updating existing shared blocks

Updating a shared block is similar, except:

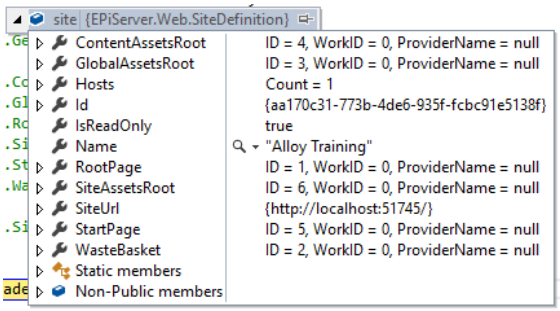
- We would only need to cast to `IContent` if we need to change the `Name` property.
- This example uses a casting expression instead of `as` keyword when we call `Save` because that method requires an object that implement `IContent`.

```
ContentReference blockLink = ...;
var repo = ServiceLocator.Current.GetInstance<IContentRepository>();
EditorialBlock editorial =
    repo.Get<BlockData>(blockLink).CreateWritableClone() as EditorialBlock;
editorial.MainBody = new XhtmlString("<p>Hello again!</p>");
repo.Save((IContent)editorial,
    EPiServer.DataAccess.SaveAction.CheckOut,
    EPiServer.Security.AccessLevel.Edit);
```


Programmatically working with sites

To get the site definition for the current request:

```
var site = SiteDefinition.Current;
```



To get a list of all sites in a multisite project:

```
var siterepo = ServiceLocator.Current
    .GetInstance<ISiteDefinitionRepository>();
IEnumerable<SiteDefinition> sites = siterepo.List();
```

ISiteDefinitionRepository methods: **List**, **Get**, **Delete**, **Save**

Module B – Working with Content using APIs – Managing content with APIs

Create, update and delete content
Deleting content

- IContentRepository methods:

```
void Delete(
          ContentReference contentLink,
          bool forceDelete,
          EPiServer.Security.AccessLevel access)

void DeleteChildren(
          ContentReference contentLink,
          bool forceDelete,
          EPiServer.Security.AccessLevel access)

void DeleteLanguageBranch(
          ContentReference contentLink,
          string languageBranch,
          EPiServer.Security.AccessLevel access)
```


Module B – Working with Content using APIs – Getting, Filtering, & Finding

Getting, filtering, finding, and searching indexed content

Type	Looks in	Notes
`IContentLoader`	Object cache, then database if necessary.	Use to programmatically generate listings and menus for navigation. Always use in combination with `FilterForVisitor` to remove unpublished, template-less, non-permissioned content.
`IPageCriteriaQueryService`	Database	Avoid, because: (1) it only finds pages, (2) it always hits the database, (3) the properties are not indexed.
`SearchHandler`	Lucene Index	Search results include content reference if you need to get the full content data.

Module B – Working with Content using APIs – Getting, Filtering, & Finding

Getting content

Type	Method	Parameter(s)	Return Type
IContentLoader	Get<T>	ContentReference	T
	GetChildren<T>	ContentReference	IEnumerable<T>
	GetAncestors<T>	ContentReference	IEnumerable<T>
	GetDescendents	ContentReference	IEnumerable<ContentReference>
	GetItems<T>	IEnumerable<ContentReference>	IEnumerable<T>

Module B – Working with Content using APIs – Getting, Filtering, & Finding

Getting content

Code defensively when getting content:

```
// throws exception if it is NOT a NewsPage
var newsPage = loader.Get<NewsPage>(contentReference);
```

```
// returns null if it is NOT a NewsPage
var newsPage = loader.Get<IContent>(contentReference) as NewsPage;
```

Filtering content

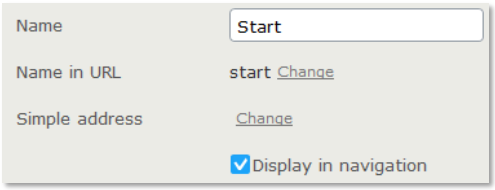
Type	Method	Parameter(s)	Return Type
FilterForVisitor	Filter	IEnumerable<IContent>	IEnumerable<IContent>
		PageDataCollection	PageDataCollection
FilterContentForVisitor	Filter	IList<IContent>	void*

Import the EPiServer.Filters namespace.

*The input parameter object will be filtered.

They filter on Access, Template and Published and has saved many developers from showing the wrong content to the wrong group/visitor.

Listing and filtering example



- Get an instance of an **IContentLoader** and get the children of the Start page.
- Apply a filter to remove (1) unpublished content, (2) content the current user shouldn't see, and (3) content without a render template, and
- Use LINQ to remove children that have their **Display in navigation** check box cleared. Note: **IContent** does not have the **VisibleInMenu** property so we must first cast back into **PageData** instances.

```
var loader1 = ServiceLocator.Current.GetInstance<IContentLoader>();
IEnumerable<PageData> childrenOfStart =
    loader1.GetChildren<PageData>(ContentReference.StartPage);
IEnumerable<IContent> filteredChildren = FilterForVisitor.Filter(childrenOfStart);
IEnumerable<PageData> displayInNavigationChilden = filteredChildren
    .Cast<PageData>().Where(p => p.VisibleInMenu);
```

Descendents and getting items example

- Get an instance of an **IContentLoader** and get the descendants of the Start page.
  Note: this would be ALL children, grand-children, great-grand-children, and so on, so the method doesn't return the **PageData** instances, instead it returns **ContentReference** instances.
- To fetch the actual **PageData** instances, **IContentLoader** has a **GetItems** method.
  Note: you must pass an instance of **LoaderOptions** as the second parameter.

```
var loader1 = ServiceLocator.Current.GetInstance<IContentLoader>();
IEnumerable<ContentReference> descOfStartAsRefs =
    loader1.GetDescendents(ContentReference.StartPage);
IEnumerable<IContent> descOfStartAsContent =
    loader1.GetItems(descOfStartAsRefs, new LoaderOptions());
```


Finding (pages only)

```
private readonly IPageCriteriaQueryService finder;
```

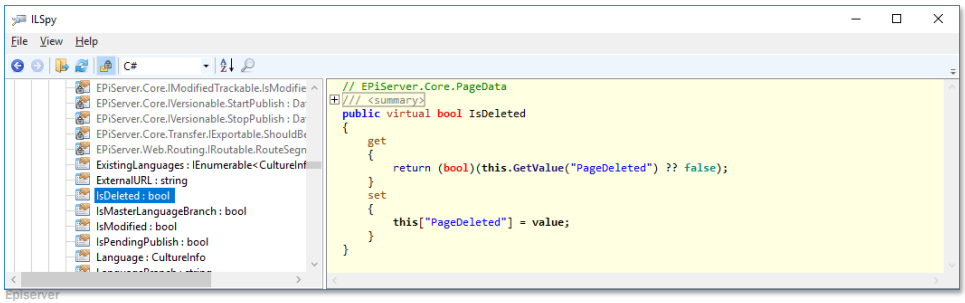
```
var criteria = new PropertyCriteriaCollection();
criteria.Add(new PropertyCriteria
{
    Type = PropertyDataType.LongString,
    Name = "PageName",
    Condition = CompareCondition.Contained,
    Value = "alloy"
});
```

```
PageDataCollection matches = finder.FindPagesWithCriteria(
    (PageReference)currentPage.ContentLink, criteria);
```

Module B – Working with Content using APIs – Getting, Filtering, & Finding

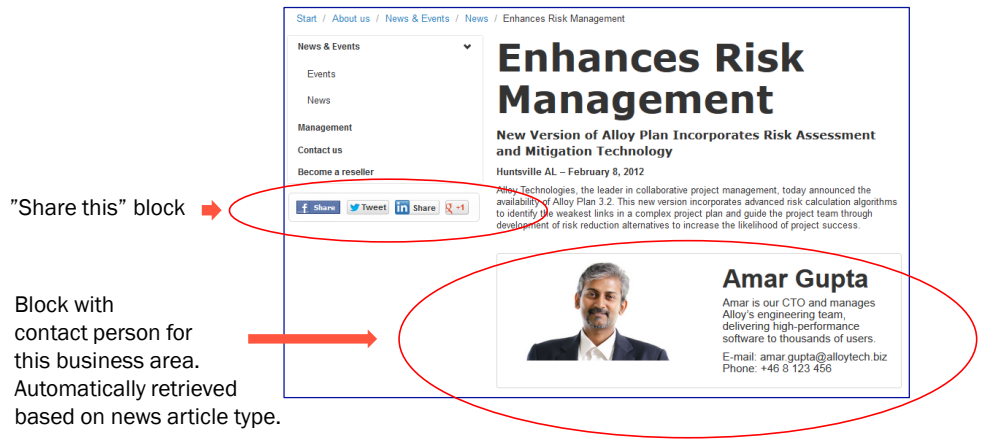
Property names when finding

When using the **FindPagesWithCriteria** method, you need to supply the name of a property using its name stored in the CMS database, not the name of a property in the class. For example, **PageData** has a property named **IsDeleted**, but you must use **PageDeleted** as the name instead. You can check what name to use by using tools like ILSpy to look inside the class implementation.



Module B – Working with Content using APIs – Sharing content

Sharing information across content



Sharing information across content: Working with shared block

- Scenario 1: Block that retrieves information from its parent content
  - A listing block that will list the child pages of its parent
  - ”Share this” block where the links need to go to the page where the block reside

```
var routeHelper = ServiceLocator.Current
    .GetInstance<EPiServer.Web.Routing.IPageRouteHelper>();
PageData myPageData = routeHelper.Page;
```


Sharing information across content: Adding shared block to content area

- Scenario 2: Add a block to a content area programmatically:

```
var contentRepository =
EPiServer.ServiceLocation.ServiceLocator.Current.GetInstance<IContentRepository>();

var page = contentRepository.Get<StandardPage>(pageReference.ToReferenceWithoutVersion(), lang);

page = page.CreateWritableClone() as StandardPage;
var ca = page.MainContentArea.CreateWritableClone();

ca.Add((IContent)newBlock);
page.MainContentArea = ca;
contentRepository.Save(page, SaveAction.Publish, AccessLevel.Publish);
```

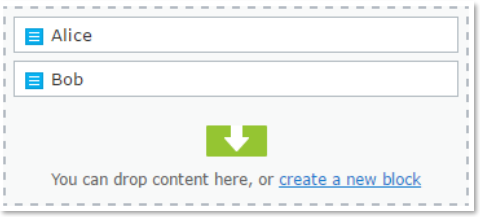
Sharing information across content: RenderContentData

```
var contentAreaItems = Model.MyContentArea.FilteredItems.Take(maxBlocksToList);
foreach (var item in contentAreaItems)
{
        var block = item.GetContent();
        Html.RenderContentData(block, true);
}
```

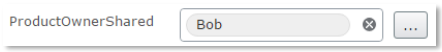
This method could for example be used if there is a need to render the items in a different way than default inside a content area or if content area items need to be displayed in a listing somewhere else.

Blocks

Shared/global blocks are created and stored in folders in the **Assets** pane and can be dragged and dropped inside a **ContentArea** property or set for a **ContentReference** property.




```
// can have references to multiple shared blocks
public virtual ContentArea PagesAndBlocks { get; set; }
// can have a reference to one shared block
[AllowedTypes(typeof(EmployeeBlock))]
public virtual ContentReference ProductOwnerShared { get; set; }
```



If a content type has a property block then it has its own instance stored as part of the content that cannot be shared.

```
// cannot reference a shared block
public virtual EmployeeBlock ProductOwner { get; set; }
```

Module B – Working with Content using APIs – Sharing content

To populate a ContentArea automatically for a new page (1 of 2)

In Alloy, the StandardPage has a MainContentArea. Imagine that it should be populated with the first block found in the **For All Sites** folder (if one exists) when a new instance is created:

```
public class StandardPage : SitePageData
{
    public override void SetDefaultValues(ContentType contentType)
    {
        base.SetDefaultValues(contentType);
        var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
        var firstBlock = loader.GetChildren<BlockData>
            (ContentReference.GlobalBlockFolder).FirstOrDefault();
```


Module B – Working with Content using APIs – Sharing content

To populate a ContentArea automatically for a new page (2 of 2)

In Alloy, the StandardPage has a MainContentArea. Imagine that it should be populated with the first block found in the **For All Sites** folder (if one exists) when a new instance is created:

```
    if (firstBlock != null)
        {
            MainContentArea = new ContentArea();
            MainContentArea.Items.Add(new ContentAreaItem
            {
                ContentLink = (firstBlock as IContent).ContentLink
            });
        }
    }
```

Module B – Working with Content using APIs – Sharing content

How to render a ContentReference

To render a `ContentReference` property in a view, you must consider what type of content it is:
If it points to a **page** or a **media** asset, and you want to render as a clickable hyperlink:

```
@Html.ContentLink(Model.MyContentReference, null,
    htmlAttributes: new { @class = "mobile" })
```

If it points to a **page** or a **media** asset or a **block**, and you want to render it using a template:

```
@using AlloyTraining.Business.ExtensionMethods
```

```
@{
    Html.RenderContentData(Model.MyContentReference.Get(),
        isContentInContentArea: false);
}
```


Module B – Working with Content using APIs – Programming content approvals

Programmatically working with Content Approvals

Get, create, update, and delete an approval sequence definition using:

- **Services**: IApprovalDefinitionRepository
- **Classes**: ContentApprovalDefinition, ApprovalDefinitionStep, ApprovalDefinitionReviewer
- **Methods**: GetAsync, SaveAsync, DeleteAsync

Work with approval workflows using:

- **Services**: IApprovalRepository, IApprovalEngine, IApprovalEngineEvents
- **Classes**: ContentApproval
- **Methods**: ApproveAsync, RejectAsync, AbortAsync, GetAsync, GetItemsAsync
- **Events**: Started, Approved, Rejected, Aborted, StepStarted, StepApproved, StepRejected

http://world.episerver.com/documentation/developer-guides/CMS/Content/content-approvals/working-with-content-approvals/

Module B – Working with Content using APIs – Programming content approvals

```
using EPiServer.Approvals;
using EPiServer.Approvals.ContentApprovals;
```

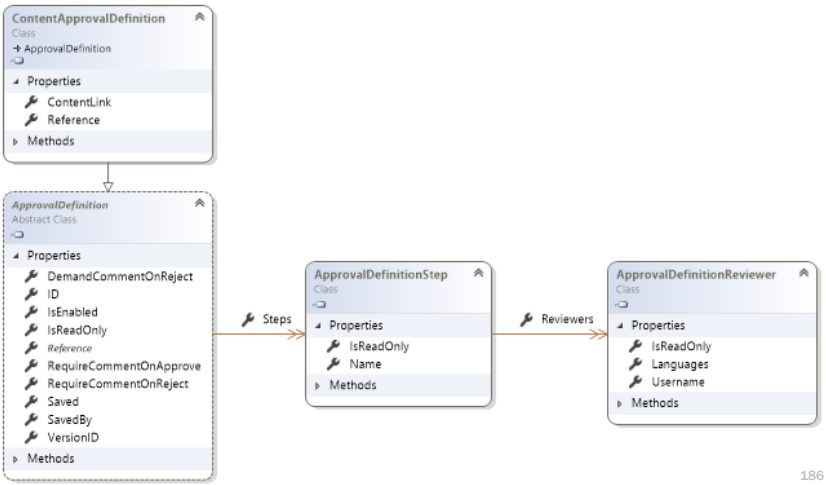
ContentApprovalDefinition

Important properties:

- **ID**
- **IsEnabled**
- **Steps**

Each step has

- **Name**
- **Reviewers**



Module B – Working with Content using APIs – Programming content approvals

```
using EPiServer;
using EPiServer.DataAccess;
using EPiServer.Security;
```

Starting the approval process

A content approval is not started by saving an *approval* but by saving a *content item* with **SaveAction.RequestApproval**. This automatically creates and saves a **ContentApproval** for this content item, if a definition can be resolved.

```
private readonly IContentRepository repoContent;
```

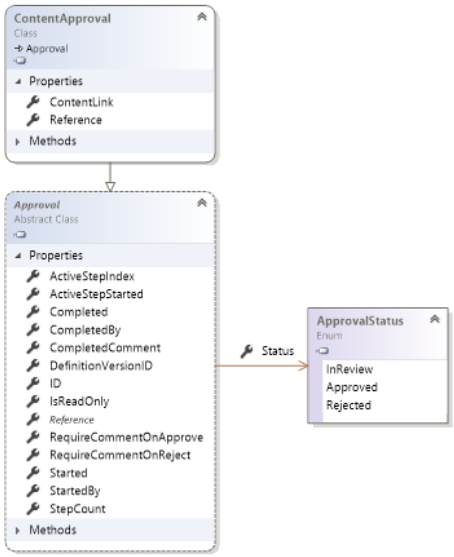
```
var start = repoContent.Get<StartPage>(ContentReference.StartPage)
    .CreateWritableClone() as StartPage;
start.Name += "X";
repoContent.Save(content: start,
    action: SaveAction.RequestApproval,
    access: AccessLevel.NoAccess);
```

ContentApproval

Each piece of content, including one for each language branch, has an instance of **ContentApproval** associated with it, once a request for approval has been made.

Important properties:

- **ID**
- **ActiveStepIndex** (0, 1, 2, and so on)
- **Status**
- **Started** (DateTime), **StartedBy**
- **Completed** (DateTime), **CompletedBy**, **CompetedComment**




```
using EPiServer.Approvals;
using EPiServer.Approvals.ContentApprovals;
```

Making a decision

Use the approval engine to decide to approve or reject/decline a step or the whole approval.

```
private readonly IApprovalRepository repoApprovals;
private readonly IApprovalEngine engine;
```

```
var approval = await repoApprovals.GetAsync(ContentReference.StartPage);
await engine.ApproveStepAsync(
    id: approval.ID,
    username: "Alice",
    stepIndex: 1,
    comment: "I approve: the page looks great!");
```

CMS users need `AccessLevel.Change` to be able to approve or decline a step.

Programmatically working with notifications

Notification API is intended for sending user-to-user notification messages.

You can create your own formatters and providers.

Every message is sent on a channel (identified by a channel name), which is a namespace that groups messages of a certain kind together.

The sender has no control of how the recipient receives the message.

Notifications are stored in the database and old notifications are deleted by the Notification Message Truncate scheduled job, which is set to run every night by default and removes all notifications older than 3 months.

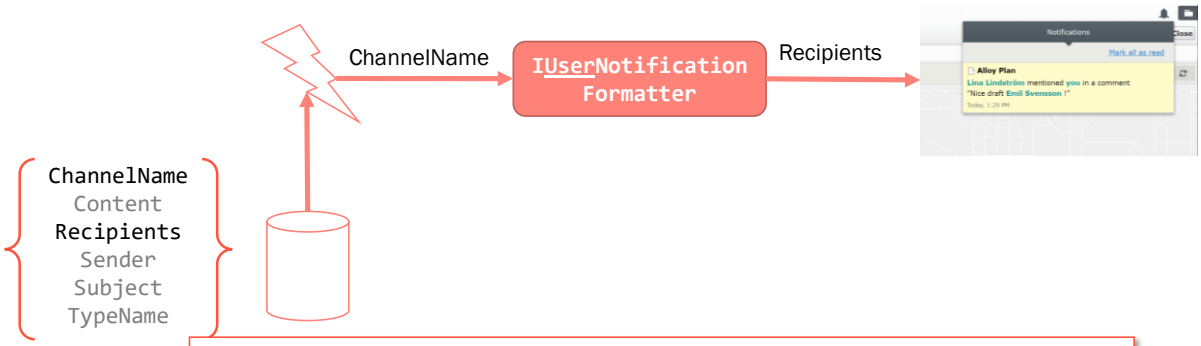
Messages are sent using INotifier

- PostNotificationAsync

http://world.episerver.com/documentation/developer-guides/CMS/using-notifications/

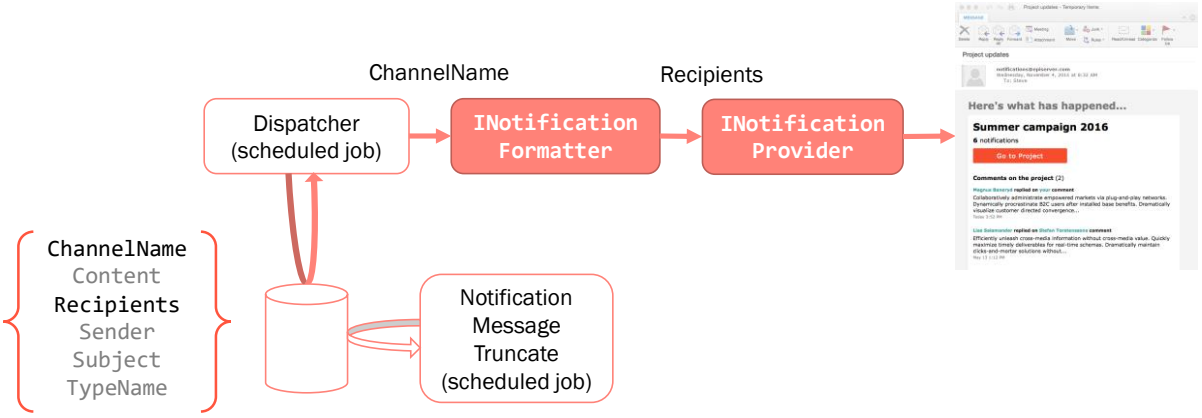
Instant user notifications

A message can either be configured to be sent immediately or be placed in a message queue that is periodically handled by a scheduled job.



http://world.episerver.com/documentation/developer-guides/CMS/using-notifications/usage-examples/

Delayed and scheduled notifications



Notification subscriptions

Subscription API supports storing a link between a key and an user. You can then later use the API to get a list of users subscribing to a key. A key can be anything you want formatted as an Uri, for example, content like a page in CMS or product in Commerce.

ISubscriptionService

- Subscribe+
- Unsubscribe+
- ClearUser+
- ClearSubscription+
- ListSubscriptions
- ListSubscribers
- FindSubscribers

http://world.episerver.com/documentation/developer-guides/CMS/using-notifications/subscription_keys/

Module B – Working with Content using APIs – Programming notifications

Notification formatters and providers

INotificationFormatter
- FormatterName
- SupportedChannelNames
- FormatMessages

IUserNotificationFormatter
- FormatUserMessage

INotificationProvider
- ProviderName
- GetProviderFormat
- Send

INotificationProviderStatus
- IsDisabled
- DisabledReason

Module B – Working with Content using APIs – Programming key performance indicators

Programmatically working with KPIs

A key performance indicator (KPI) in Episerver records when a visitor on a website performs a desired action, such as clicking on an add button, or completing a sale.

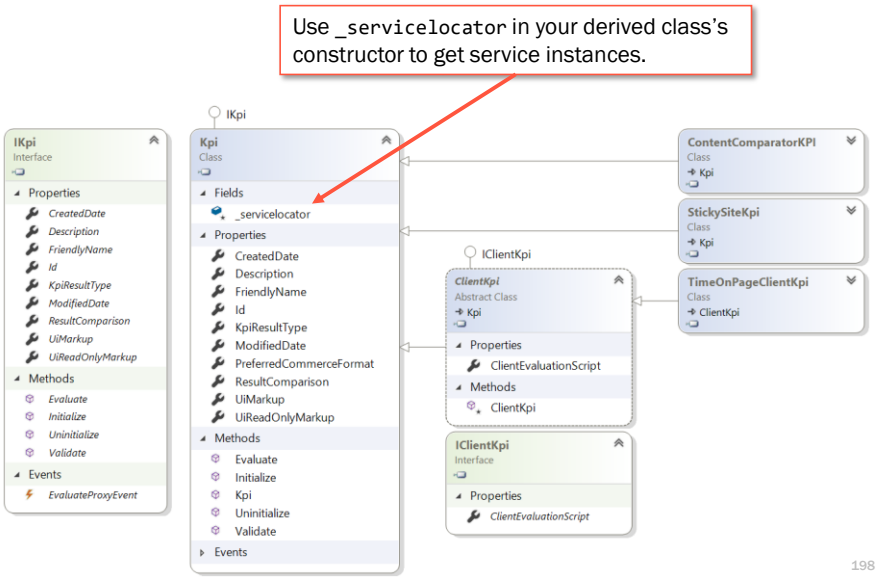
KPIs can be used as conversion goals in A/B testing.

`IClientKpi` is an interface for marking a custom KPI that should be run on the client browser to convert an A/B test. It consists of only one method named `ClientEvaluationScript` for retrieving the client JavaScript that needs to be presented in the browser to indicate when a conversion takes place.

http://world.episerver.com/documentation/developer-guides/CMS/key-performance-indicators-kpis/

Implementing a KPI

To create a custom KPI, implement the `IKpi` interface (server-side evaluation) and optionally the `IClientKpi` interface (client-side evaluation), as the three builtin KPIs for CMS A/B testing do, as shown in this class diagram.

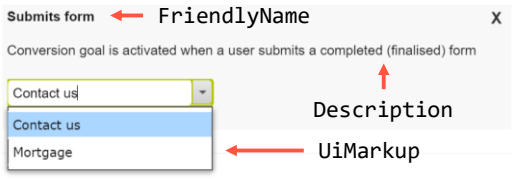


Setting up inputs for a conversion goal

When an editor creates an A/B test, and they choose your custom conversion goal, you control the user experience via some properties of `IKpi`:

- `FriendlyName` and `Description`: strings to name and describe the goal in the user interface.
- `UiMarkup`: returns a string of HTML for any custom inputs your goal needs, like a form selection.

To check a correct input has been made, implement the `Validate` method. You will be passed a dictionary of string values for all the inputs. Throw a `KpiValidationException` if there is a problem.

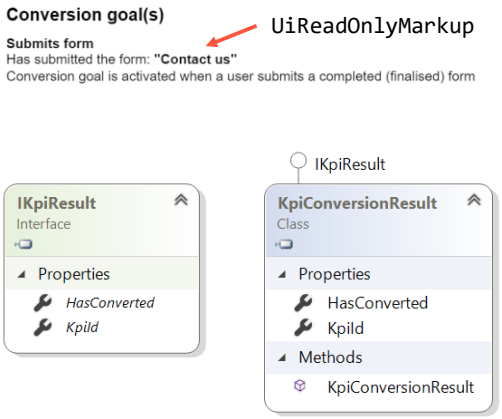


Running and evaluating a test with a custom conversion goal

Once a test is running, the implementation of `UiReadOnlyMarkup` is used to show the inputs of the custom conversion goal.

Implement the `EvaluateProxyEvent` event to add and remove a handler for the CMS content event that will trigger the conversion goal.

Implement the `Evaluate` method to return an `IKpiResult` with its `HasConverted` property set to `true` if a conversion has been made.



Some other repositories

Working with languages:
- ContentLanguageSetting
  - (e.g. language selector)
- LanguageBranch
  - (get active language)
- LanguageBranchRepository
- ILanguageBranchRepository

Working with unpublished versions of content:
- IContentVersionRepository
- ContentVersion
- IContentSecurityRepository

Working with MVC:
- TemplateModel
- TemplateModelRepository

UI and content type definitions:
- TabDefinitionRepository
- TabDefinition
- PageType/PageTypeRepository
- ContentType
- BlockType
- PropertyDefinition
- PropertyDefinitionRepository
- BackingTypeResolver

Module B – Working with Content using APIs – Other APIs

Programmatically working with Projects

Get, create, update, and delete a projects and their content items using:

- **Types**: ProjectRepository, Project, ProjectItem, ProjectResolver
- **Methods**: Save, SaveItems, Get, GetItem, List, ListItems, FindItems, GetCurrentProjects, Delete, DeleteItems
- **Events**: ProjectSaved, ProjectDeleted, ProjectItemsSaved, ProjectItemsDeleted

Publish projects using:

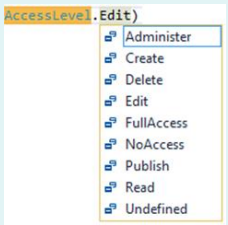
- **Types**: ProjectPublisher
- **Methods**: PublishAsync, ReactivateAsync

http://world.episerver.com/documentation/developer-guides/CMS/projects/creating-a-project-programmatically/

Module B – Working with Content using APIs – Other APIs

Managing access rights programmatically

```
contentRepository.Save(updatedContent,
                       EPiServer.DataAccess.SaveAction.Publish,
                       EPiServer.Security.AccessLevel.NoAccess);
```




```
[Access(Roles = "News_Editors", Users = "user1, user2")]
public class NewsPage : PageData {}
```

Update/Read ACL (Access Control List) from code

```
using EPiServer.Security;
```

- Read:

```
ContentAccessControlList cacl = new ContentAccessControlList(page.ContentLink);
AccessControlList acl = page.ACL;
```

for specific page

```
AccessLevel al = CurrentPage.ACL.QueryAccess();
bool hasEditAccess = CurrentPage.ACL.QueryDistinctAccess(AccessLevel.Edit);
```

for current user

- Update:

```
AccessControlList acl = page.ACL.CreateWritableClone();
acl.Clear();
acl.Add(new AccessControlEntry("BlogOwnerGroup", AccessLevel.Read, SecurityEntityType.Role));
acl.Save();
```

- Get notified when a content's ACL has changed:

```
ServiceLocator.Current.GetInstance<EPiServer.DataAbstraction<IContentSecurityRepository>().ContentSecuritySaved +=
new EventHandler<ContentSecurityEventArg>(DoSomethingWhenContentSecuritySaved);
```


Checking page languages from code

- Get all language branches with content repository

```
PageDataCollection allLanguages = contentRepository.GetLanguageBranches(myPageReference);
```

- Check if page exists in specific language

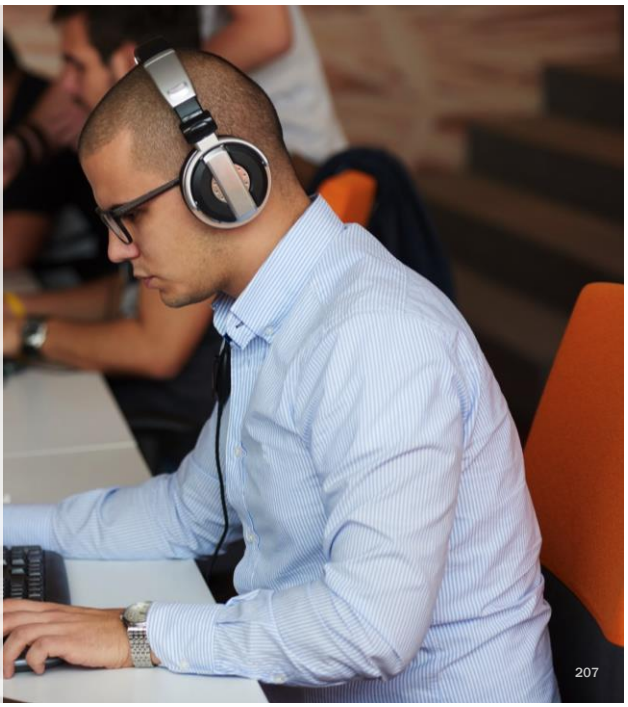
```
bool exists = contentRepository
        .GetLanguageBranches(PageReference.StartPage)
        .Any(p => p.LanguageBranch == "no");
```

- Or use the PageData.ExistingLanguages property that returns an array of CultureInfo instances.

Module B – Working with Content using APIs

Exercises for Module B

1. Implementing a Share This Page block
2. Programming content approvals
3. Implementing notifications
4. Implementing a commenting solution



Episerver

207

Episerver CMS Advanced Development

Module C
Integrating Data and Forms

An Episerver site can contain content that does not need to have all the functionality that regular editorial content has, such as versions, scheduling, etc. You can choose to save it to the Dynamic Data Store, or you may need to integrate an external data store.

Episerver

208

Module C – Integrating Data and Forms

Topics

- Overview
  - Data integration choices
- Forms
  - Episerver CMS XForms
  - Episerver Forms
- Scheduled jobs and content events
- Episerver Service API
- Partial routing
- Dynamic Data Store

Module C – Integrating Data and Forms – Overview

Overview

Most of the data used by the Episerver CMS is *managed content* that can be:

- Easily created and managed,
- Localized into languages like English, Spanish, and Swedish,
- Have access rights applied, and
- Have versions and publishing workflow.

Some of the data used by Episerver CMS sites does not need those features, or it is stored outside the CMS database so it cannot support those features. But we might still want it to appear as part of a single system. We might want to gather data from a visitor using forms defined by editors.

In this module you will learn about your options for handling non-content data.

Module C – Integrating Data and Forms – Overview

Data integration choices

Technology	Direction	Live?	Description	Effort
Partial Router	One-way, read-only	Live	URL path that pulls data from an external system.	Medium
Content Provider	Two-way, read-write	Live	Manage content stored in an external system. The CMS itself is the default *content provider*.	Large
Scheduled Job	Two-way, read-write	Scheduled	Custom import/export data to/from an external system. Also use Initialization Module with IContentEvents for live sync.	Small
Dynamic Data Store	Two-way, read-write	Live	Custom storage of any .NET type. Performance can be poor.	Small
Service API	Varies	Live	REST API for integration with external systems.	Medium

http://world.episerver.com/documentation/developer-guides/CMS/Content/Content-providers/

Module C – Integrating Data and Forms – Forms

Episerver CMS XForms

XForms is Episerver's older forms technology. In CMS 11 it must be installed as a separate package.

- XForms is based on the W3C standard.
- In Edit view, an `XForm` property provides a list of XForms that can be selected and the ability to define new ones.
- The form layout uses HTML tables which is poor practice for modern responsive design.
- XForms data is stored in Dynamic Data Store (DDS) as XML.

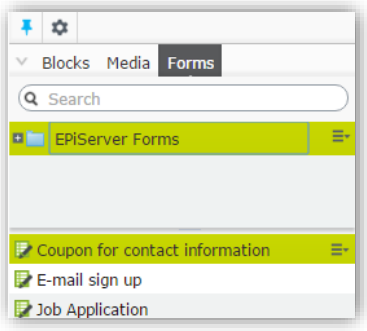
http://world.episerver.com/documentation/developer-guides/CMS/forms/xforms-legacy-functionality/

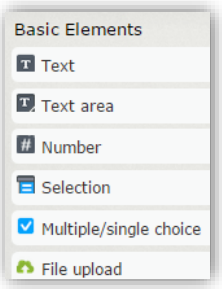
Reasons to use XForms:

- You are using Episerver CMS 8 or earlier.
- You are using ASP.NET Web Forms.

Module C – Integrating Data and Forms – Forms

Episerver Forms





Episerver Forms is a newer forms technology distributed as a NuGet package. The user interface is accessed as a plug-in for the Asset pane in Edit view.

Episerver Forms is only supported in ASP.NET MVC sites with Episerver CMS 9 or later.

Episerver Forms form definitions are stored as content in the CMS, in a similar way to blocks, so they can support all features of content such as versions and localization.

Episerver Forms data is stored in Dynamic Data Store (DDS).

http://world.episerver.com/documentation/developer-guides/forms/

http://world.episerver.com/add-ons/episerver-forms/

http://world.episerver.com/blogs/Allan-Thran/Dates/2015/11/introducing-episerver-forms/

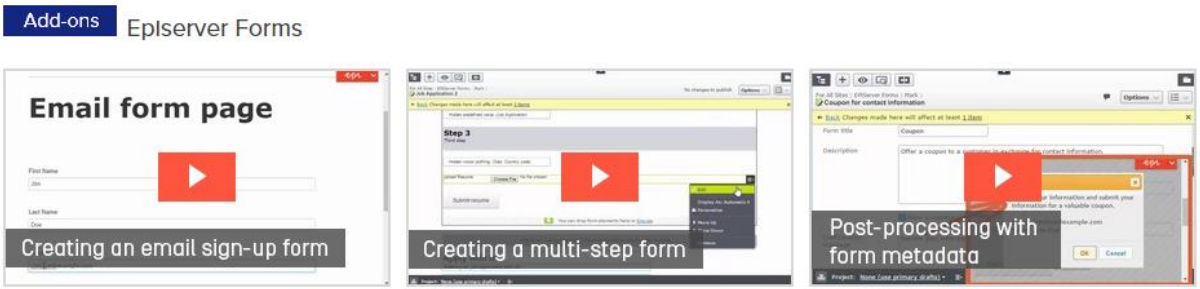
Episerver 215

Module C – Integrating Data and Forms – Forms

Episerver Forms videos

Use the following link and scroll down to the Episerver Forms section:

http://webhelp.episerver.com/latest/_online-only-topics/videos.htm



Episerver 216

Styling Episerver Forms

The built-in form elements have minimal styling rules with the expectation that a developer will modify the appearance for site application.

You can alter the default styling of a form by directly modifying the CSS file in `wwwroot\modules\_protected\EPiServer.Forms\0.22.0.9000\ClientResources\ViewMode`

http://world.episerver.com/documentation/developer-guides/forms/css-styling-and-javascript/

Episerver Forms encryption

By default, form submission data is stored as plain text. However, in some environments, the law requires the encryption of that data.

Episerver Forms can use **Azure KeyVault** to store the Advanced Encryption Standard (AES) symmetric algorithm key. Episerver then retrieves the key from KeyVault and uses it for encryption and decryption.

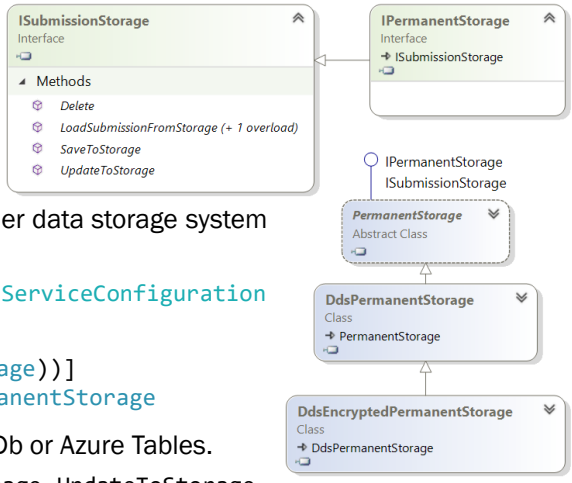
To enable Episerver Forms encryption:

1. Create a **secret** in Azure KeyVault.
2. Install the Nuget package **EPiServer.Forms.Crypto.AzureKeyVault**
3. Enable session state.
4. Modify the storage provider configured in the `~/modules/_protected/EPiServer.Forms/Forms.config` file, as described at the following link:

   http://world.episerver.com/documentation/developer-guides/forms/encrypting-form-data/

Module C – Integrating Data and Forms – Forms

Episerver Forms custom storage mechanism



You can replace Dynamic Data Store (DDS) with another data storage system for visitor submissions.

1. Inherit from `PermanentStorage` and decorate with `ServiceConfiguration` attribute:

```
[ServiceConfiguration(typeof(IPermanentStorage))]
public class MongoDbPermanentStorage : PermanentStorage
```

2. Set up your storage provider, for example, MongoDb or Azure Tables.
3. Override and implement the methods: `SaveToStorage`, `UpdateToStorage`, `LoadSubmissionFromStorage` (two overloads), and `Delete`.

https://world.episerver.com/documentation/developer-guides/forms/creating-new-data-storage-mechanism/

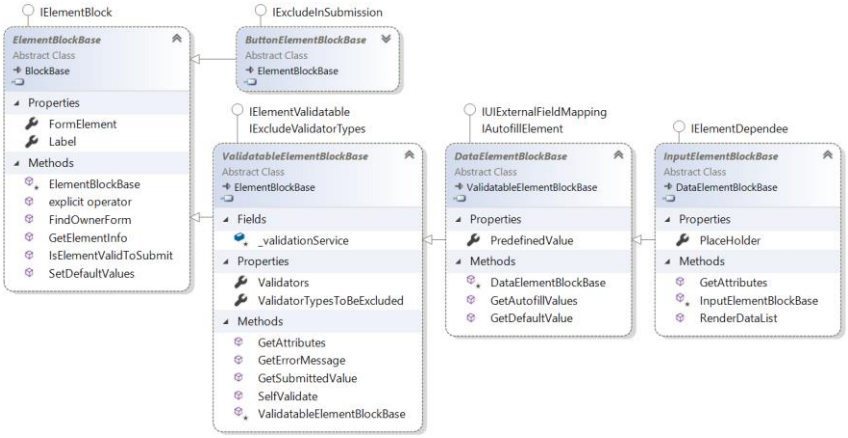
Episerver 219

Module C – Integrating Data and Forms – Forms

Episerver Forms custom form elements

`ElementBlockBase` class is the only type allowed in the form container's content area.

Inherit from `ElementBlockBase`–derived classes to support validation.



Episerver 220

Module C – Integrating Data and Forms – Forms

Episerver Forms custom actors

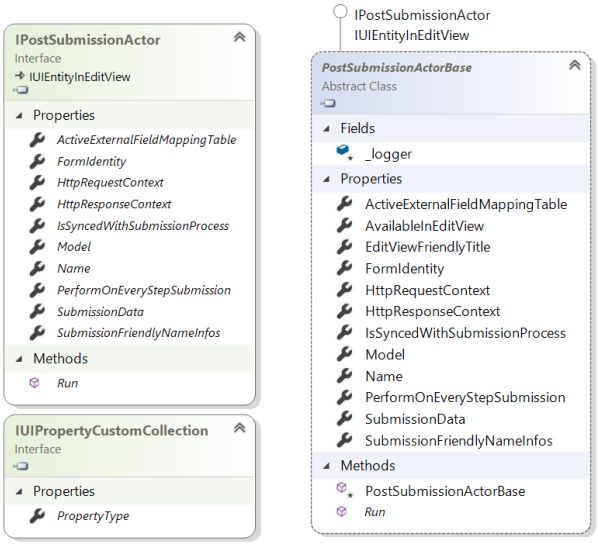
Episerver Forms include two built-in actors: `SendEmailAfterSubmissionActor` and `CallWebhookAfterSubmissionActor`.

To define a custom actor you should:

- Inherit from `PostSubmissionActorBase` base class or implement the `IPostSubmissionActor` interface.
- Implement `IUIPropertyCustomCollection` interface.

Implementing a custom Actor

https://world.episerver.com/documentation/developer-guides/forms/implementing-a-customized-actor/



Module C – Integrating Data and Forms – Scheduled jobs and content events

Integrating with scheduled jobs and content events

Instead of using the built-in integration methods you can build your own custom integration with scheduled jobs using the full capabilities in the .NET Framework. For example, a scheduled job that reads external data in any format, and inserts it in the CMS using Episerver content repository.

Scheduled jobs are often preferred to content providers and content channels because it is easier to work with:

- The developer has more control
- Less development is needed in comparison

http://world.episerver.com/documentation/developer-guides/CMS/scheduled-jobs/

Scheduled jobs and multiple sites and servers

If several sites share a database, such as in a load-balanced scenario, you can control which site executes scheduled jobs.

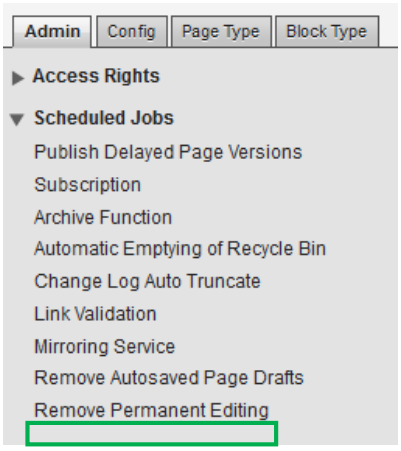
- Set the **enableScheduler** attribute to **true** on the **applicationSettings** configuration element on the site that should execute the jobs, and to **false** on the other sites.

If you configure several sites to run scheduled jobs, each job is scheduled for execution on all sites. However, during execution, the first site that starts executing a job marks it in the database as executing, so the other sites do not execute that job in parallel.

Creating scheduled jobs

```
[ScheduledPlugIn(DisplayName = "Simulated Job")]
public class SimulatedScheduledJob : ScheduledJobBase
{
    private bool _stopSignaled;

    public SimulatedScheduledJob()
    {
        IsStoppable = true;
    }

    public override void Stop()
    {
        _stopSignaled = true;
    }
```

Unhandled exceptions are automatically caught and returned to the user interface as a “failed” job.



Implementing scheduled job's execution (1 of 2)

Add a new project item of type **Scheduled Job** and implement its **Execute** method as shown below:

```
public override string Execute()
{
    // if this job is run manually then this will NOT be null and the current user
    // permissions will be checked, else, we might need to assign higher permissions.
    if (HttpContext.Current == null)
    {
        PrincipalInfo.CurrentPrincipal = new GenericPrincipal(
            new GenericIdentity("Scheduled Job Demo"),
            new[] { "Administrators" });
    }
    OnStatusChanged(string.Format("Starting execution of {0}", GetType()));
    var r = new Random();
    int percentComplete = 0;
```

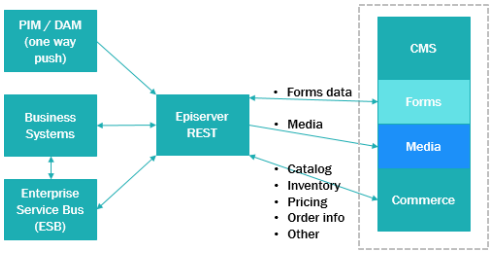

Implementing a scheduled job's execution (2 of 2)

Finish the implementation of **Execute** method, and then execute the job from Admin view.

```
    while (percentComplete < 100)
    {
        System.Threading.Thread.Sleep(2000);
        percentComplete += r.Next(5, 15);
        OnStatusChanged(string.Format(
            "{0}% complete. Please wait...", percentComplete));
        if (_stopSignaled)
        {
            return "Stop of job was called";
        }
    }
    return "Completed successfully!";
}
```


Episerver Service API

Service API is a service layer available for system integrators to update and retrieve information from Episerver, ensuring a seamless integration with external systems such as PIM, DAM and ERP.



Service API provides a programming interface for performing operations like:

- Import and export of "episerverdata" files, Episerver Forms data, and media and catalog data in Commerce.
- Bulk asset linking between media and catalog content in Commerce.
- "RESTful" CRUD operations for managing individual catalogs, nodes, entries, and warehouses in Commerce.
- Video: http://fast.wistia.net/embed/iframe/3ggaanph3f?videoFoam=true

CMS content import/export service URLs

- CMS site bulk import with file
  `episerverapi/commerce/import/cms/site/{siteName}/{hostname}/{culture=}`
- CMS site bulk import with file upload identifier
  `episerverapi/commerce/import/cms/site/{siteName}/{hostname}/{uploadId:guid}/{culture=}`
- CMS assets bulk import with file
  `episerverapi/commerce/import/cms/assetglobalroot`
- CMS site bulk import with file upload identifier
  `episerverapi/commerce/import/cms/assetglobalroot/{uploadId:guid}`
- CMS bulk export
  `episerverapi/commerce/export/site/{siteName}`

https://world.episerver.com/documentation/developer-guides/Episerver-Service-API/working-with-bulk-operations-using-tasks/cms-content-import-service/

Partial routing

- Makes it possible to extend routing beyond pages
  - You can use partial routing either to link to data outside Episerver CMS or to link to other content types than pages
  - In Episerver Commerce, partial routing is used for presenting catalog data for editing in the new Catalog editing interface, as well as for all product/catalog references.
  - A site for a product company were suppliers are listed as links on the start page but where the supplier-pages are container pages. A Supplier Landing Page lists the suppliers in a default order and shows picture and description from the container page. the customer wants that when someone clicks on a supplier link on the start page, the landing page will be displayed with the selected supplier first in the list. This can be solved with routing.

Implementing IPartialRouter<TContent, TRoutedData>

Each partial router must implement the interface `IPartialRouter<TContent, TRoutedData>` in the `EPiServer.Web.Routing` namespace.

The interface contains the following methods:

- `RoutePartial`
  Called when the ordinary page routing has routed to a page of type `TContent` and there is a remaining part of the URL. The implementation can then route the remaining part of the URL.
- `GetPartialVirtualPath`
  Called when an outgoing URL is constructed for a content instance of type `TRoutedData`.

Module C – Integrating Data and Forms – Dynamic Data Store

Introduction to Dynamic Data Store (DDS)

Offers an API and infrastructure for the saving, loading and searching of both compile time data types (.NET object instances) and runtime data types (property bags).

- Dynamic Data Store has been specifically designed with Episerver CMS and its flexible user driven data in mind.
- Example of data to store:
  - Comments, Page view statistics, visitor group statistics
- Video: http://fast.wistia.net/embed/iframe/pw7ebt2st1?videoFoam=true

Module C – Integrating Data and Forms – Dynamic Data Store

Core concepts

- tblBigTable: contains many columns, several of each data type supported by DDS.
- OR-mapper.
- EPiServer.Data namespace in EPiServer.Data assembly.
- Use LINQ to retrieve data from DDS.
- Alternative technologies to DDS include Microsoft's Entity Framework and NHibernate for .NET.
- Dynamic Data Store has been specifically designed with Episerver CMS and its flexible user driven data in mind.

EPiServer.Data assembly

- **EPiServer.Data** namespace
  - contains important classes used by in the Dynamic Data Store, most notably the Identity class.
- **EPiServer.Data** configuration
  - contains the configuration classes for the Dynamic Data Store.
- **EPiServer.Data.Dynamic** namespace
  - contains the DynamicDataStoreFactory and DynamicDataStore classes as well as their support classes and data structures.
- **EPiServer.Data.Dynamic.Providers** namespace
  - contains the SqlServerDataStoreProvider class as well as their base classes and other database specific classes for LINQ support.

Inline mapping

Inline mapping is where a property of a class or PropertyBag can be mapped directly against one of the supported “big table” database columns. The following types can be mapped inline:

- System.Byte (and arrays of), System.Int16, System.Int32, System.Int64, System.Enum, System.Single, System.Double, System.DateTime, System.String, System.Char (and arrays of), System.Boolean, System.Guid, EPiServer.Data.Identity

All properties that cannot be mapped inline or as a collection (plus the EPiServer.Data.Dynamic.PropertyBag type) are mapped as references.
This means that their properties are mapped in-turn as a sub-type and a link row is added in the reference table to link the parent data structure with the child data structure.
This allows for complex trees of data structures (object graphs) to be saved in the Dynamic Data Store at the cost of low performance.

tblBigTable schema – Mandatory columns

- **pkId** is the store ID and primary key of each data structure stored.
- **Row** is the row index. Each structure may span 1 or more rows in the “big table”.
- **StoreName** is the name of the store the data structure belongs to.
- **ItemType** is the .NET CLR Type that contained the properties saved to the current row.

tblBigTable schema – Optional columns

- BooleanXX (where XX is 01 through to 05) x 5
- IntegerXX (where XX is 01 through to 10) x 10
- DateTimeXX (where XX is 01 through to 05) x 5
- StringXX (where XX is 01 through to 10) x 10
- And so on
- Indexed_Boolean01
- Indexed_IntegerXX (where XX is 01 through to 03) x 3
- And so on

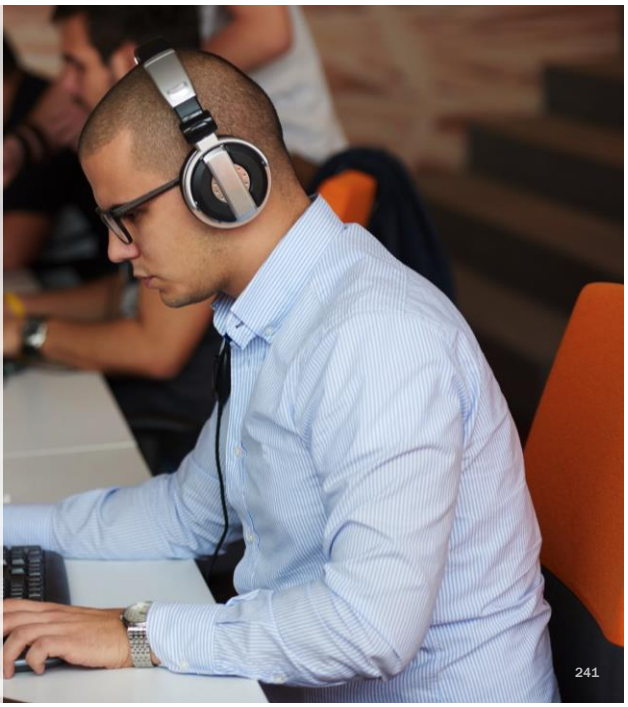
The columns whose name starts with “Indexed” have database indexes created on them.

You can add as many of each column as you like, but make sure that you write a SQL script to do that and then execute it during deployment.

Module C – Integrating Data and Forms

Exercises for Module C

1. Implementing favorite pages with Dynamic Data Store
2. Integrating external data using a Partial Route
3. Gathering data using Episerver Forms
4. Importing data using a Scheduled Job



Episerver

241

Episerver CMS Advanced Development

Module D
Customizing Properties for Editors

Properties are central in Episerver and something that the editor uses daily. Common editor tasks can often be solved, given that you as a developer know how the Episerver property works and how it can be modified.

Episerver

242

Module D – Customizing Properties for Editors

Topics

- Overview
- Content type synchronization process
- TinyMCE
- Property types
- Property validation
- UIHints and EditorDescriptors
- Dojo
- On-Page Editing

Module D – Customizing Properties for Editors – Overview

Content types and properties

- Content type
  - Defines a type of content
    - Type of page: start page, article page, search page
    - Type of block: teaser block, editorial block, contact block
  - Strongly typed, handled from code
- Properties
  - A type of content can contain a set of properties
  - Property on the model class
  - Values are stored in database
  - Editable from edit view and versioned in Episerver

Module D – Customizing Properties for Editors – Content type synchronization process

What happens when a type is registered?

If you have an Empty Episerver site, add the following class, and start the site, what happens?
http://world.episerver.com/documentation/developer-guides/CMS/Content/Synchronization/

```
namespace EmptySite.Models.Pages
{
    [ContentType(DisplayName = "Start",
        GUID = "023964e5-9df2-4fa2-8434-7ccc20e5c4b8",
        Description = "The site's home page.")]
    public class StartPage : PageData
    {
        public virtual int Age { get; set; }
    }
}
```


Module D – Customizing Properties for Editors – Content type synchronization process

What happens when a type is registered?

dbo.tblContentType [Data] | dbo.tblPropertyDefinition [Data] | dbo.tblPropertyDefinitionType [Data]

Max Rows: 1000

pkID	ModelType	Name
1	NULL	SysRoot
2	NULL	SysRecycleBin
3	EPiServer.Core.ContentFolder,EPiServer	SysContentFolder
4	EPiServer.Core.ContentAssetFolder,EPiServer	SysContentAssetFolder
5	EmptySite.Models.Pages.StartPage, EmptySite, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null	StartPage
NULL	NULL	NULL

dbo.tblContentType [Data] | dbo.tblPropertyDefinition [Data]

Max Rows: 1000

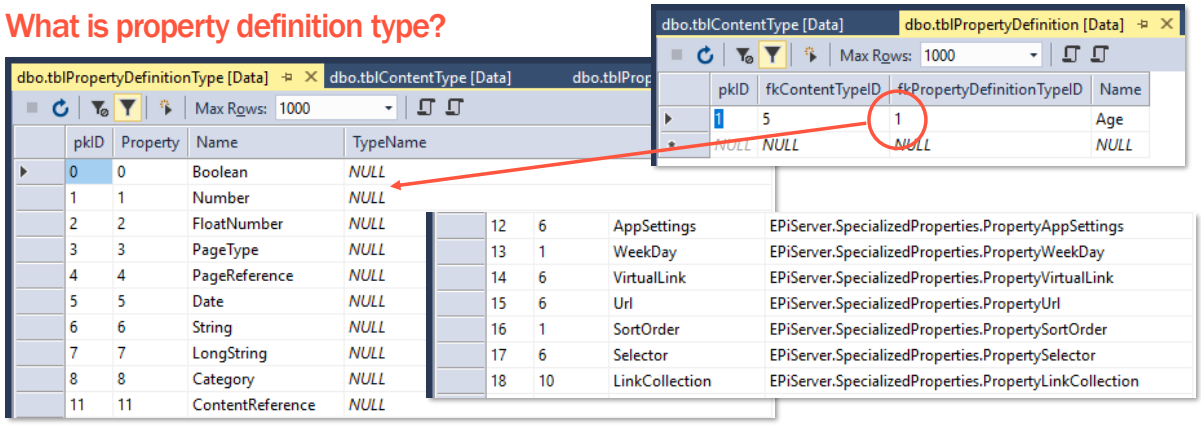
pkID	fkContentTypeID	fkPropertyDefinitionTypeID	Name
1	5	1	Age
NULL	NULL	NULL	NULL

```
public class StartPage : PageData
```

```
public virtual int Age { get; set; }
```

Module D – Customizing Properties for Editors – Content type synchronization process

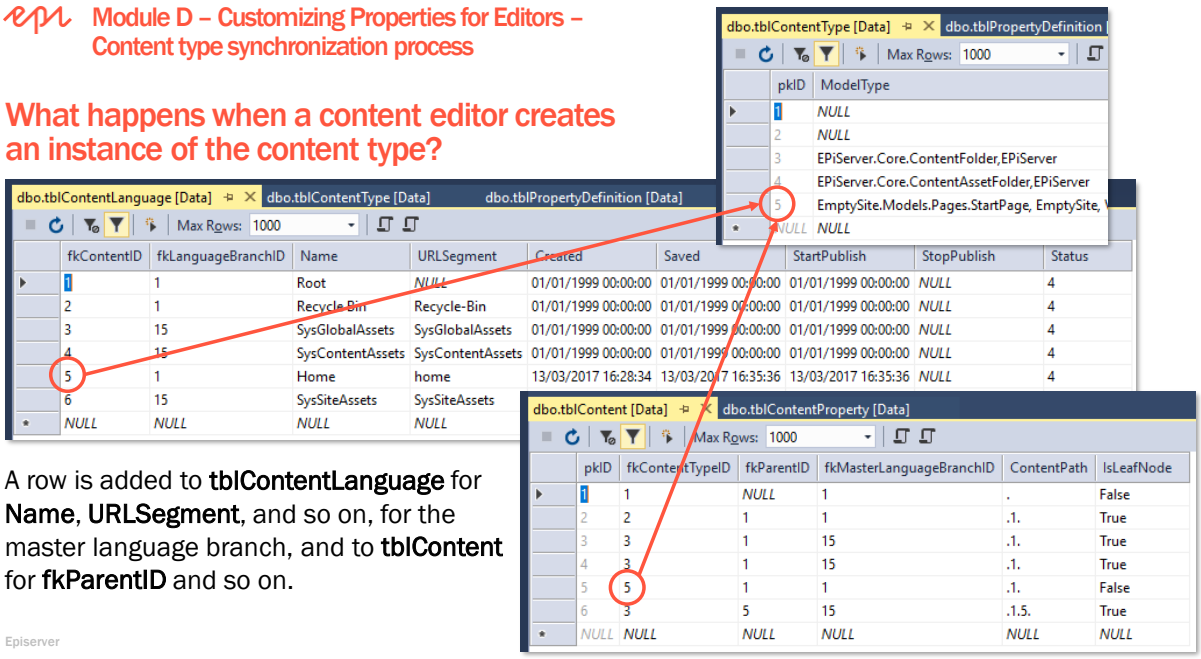
What is property definition type?



Property types like **AppSettings** and **Url** are stored as **String** types.

Module D – Customizing Properties for Editors – Content type synchronization process

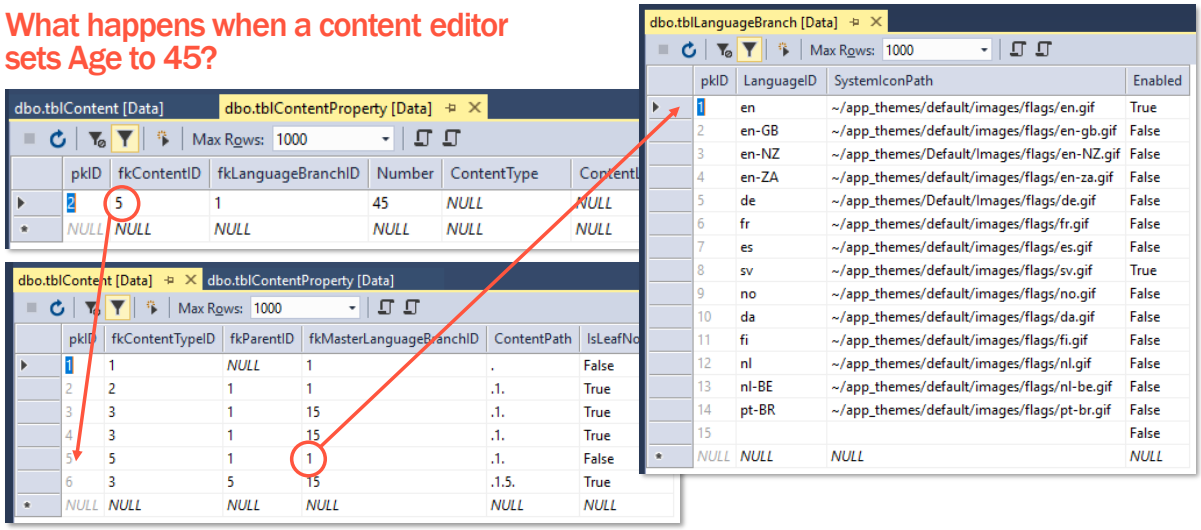
What happens when a content editor creates an instance of the content type?



A row is added to **tblContentLanguage** for **Name**, **URLSegment**, and so on, for the master language branch, and to **tblContent** for **fkParentID** and so on.

Module D – Customizing Properties for Editors – Content type synchronization process

What happens when a content editor sets Age to 45?



Module D – Customizing Properties for Editors – TinyMCE

Property type example

- Add a property to model class
  - of type EPiServer.Core.XhtmlString

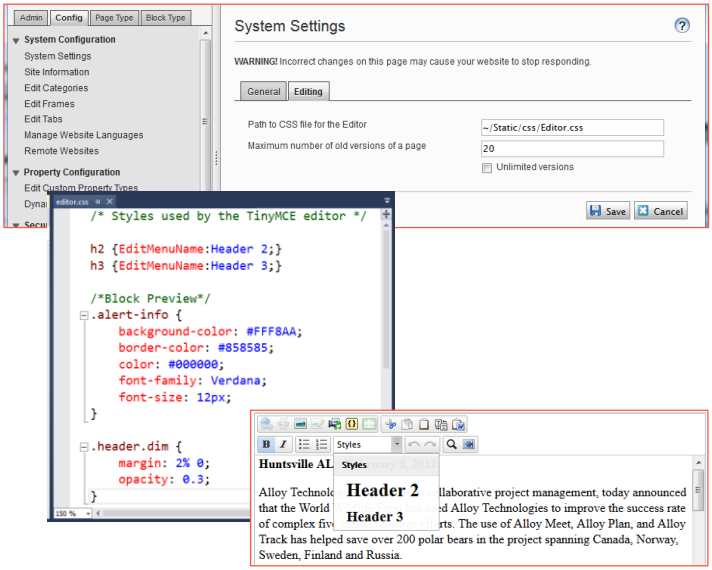
```
[Display(GroupName = SystemTabNames.Content, Order = 310)]
[CultureSpecific]
public virtual XhtmlString MainBody { get; set; }
```

- Mapped to Episerver property
  - BackingTypeResolver maps this type to EPiServer.Specialized.PropertyXhtmlString
  - It is edited using the TinyMCE editor and handles personalized content and permanent links.

Module D – Customizing Properties for Editors – TinyMCE

TinyMCE – Customizing styles

Adding styles to TinyMCE and translating the style names to the languages the UI is used in is very important. It will increase the quality of the editing interface and also of the content itself a lot because it gives the editors the possibility to work with the styles and get an immediate realistic preview of the content instead of having to jump between edit view and preview.



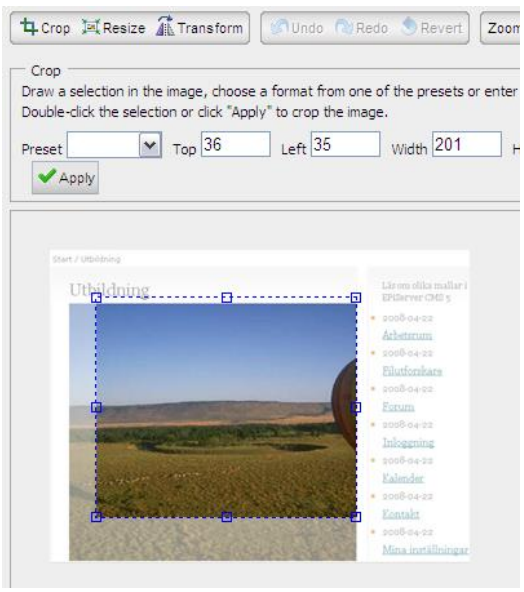
Module D – Customizing Properties for Editors – TinyMCE

TinyMCE – Customizing the image editor

With the image editor there are functions to crop and resize images. By using preset formats you can make it easier for editors to resize and crop to most commonly used sizes on the web site.

To customize the preset sizes, modify the imageEditor section in Web.config:

```
<episerver>
  <imageEditor>
    <sizePresets>
      <preset width="250" height="150" />
      <preset width="150" height="250" />
```



Property types

- BackingTypeResolver
  - Determines how data is saved to the DB
  - http://blog.fredrikhaglund.se/blog/2013/02/16/episerver-cms-7-backingtyperesolver/
- EPiServer.SpecializedProperties
  - Contains built-in special property types

Backing type	Type	UI Hint
PropertyContentArea	ContentArea	
PropertyBoolean	Boolean	
PropertyCategory	CategoryList	
PropertyContentReference	ContentReference	Many different options
PropertyDate	DateTime	Range
PropertyFloatNumber	Double	Range
PropertyLinkCollection	LinkItemCollection	
PropertyNumber	Byte	
	Int32	Range
PropertyPageType	PageType	
PropertyLongString	String	Textarea StringLength
PropertyUrl	Url	Image Document Video
PropertyXForm	XForm	
PropertyXhtmlString	XHtmlString	
PropertyBlob	Blob	
PropertyContentReferenceList	IList<ContentReference>	AllowedTypes

Content type property types

You can only use a limited number of .NET types for properties in Episerver:

- Value types should be nullable (except bool): **bool**, **int?**, **double?**, **DateTime?**, and so on.
- Reference types: **string**, **XhtmlString**, **PageReference**, **ContentReference**, **LinkItemCollection**, **Url**, **XForm**, **IList<ContentReference>**

If you want to use an unsupported type, and that type can be converted into a simpler type, for example enums can be converted into integers and strings, then you can apply the [BackingType] attribute to specify how to store and type in the CMS database:

```
[BackingType(typeof(PropertyNumber))]
[UIHint("SortOrder")]
[DefaultValue(FilterSortOrder.PublishedDescending)]
public virtual FilterSortOrder SortOrder { get; set; }
```

Recommended: make value types nullable except bool.

Property types using .NET types

.NET Type	Purpose	Examples of common attributes

`string`	Textual values in text box or text area of varying lengths and matching a pattern.	`[StringLength(50, MinimumLength = 5)]`
		`[RegularExpression("[a-zA-Z]+")]`
		`[UIHint(UIHint.Textarea)]`
		`[SelectOne(…)]`
		`[SelectMany(…)]`

`bool`	True/false values with check box editor.	

`int?`
`byte?`
	Whole number value or enum value.	`[Range(18, 65)]`
		`[SelectOne(…)]`
		`[SelectMany(…)]`

`DateTime?`	Date and time value with graphical picker.	
`double?`	Floating point value.	

Property types using Episerver types

Episerver Type	Purpose	Examples of common attributes
`XhtmlString`	Rich text, image media, and blocks.	
`Url`	A link to a page, media, email, or external URL, including query string parameters.	`// show file type-specific UI` `[UIHint(UIHint.Image/Video/MediaFile)]`
`LinkItemCollection`	A collection of links to pages, media, emails, or external URLs.	
`XForm`	Enables selection and management of XForms.	

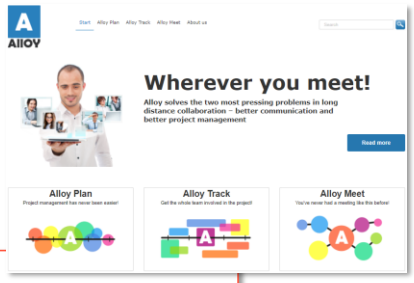
Property types using Episerver content reference types

Episerver Type	Purpose	Examples of common attributes for all these types

`ContentReference` or `PageReference`	A reference to one item of content or one page.	`// allow standard pages`
		`// but not product pages`
		`[AllowedTypes(typeof(StandardPage),`
		`RestrictedTypes =`
		`new[] { typeof(ProductPage) })]`
		
		`// allow blocks and employee pages`
		`[AllowedTypes(typeof(BlockData),`
		`typeof(EmployeePage))]`
		
		`// show file type-specific UI`
		`[UIHint(UIHint.Image/Video/MediaFile)]`

`ContentArea`	An ordered collection of references to blocks and pages (rendered using their partial template).	
`IList<ContentReference>`	A list of references to content.	

You cannot have an `IList<PageReference>`.



How are ContentAreas stored?

The references to content in a `ContentArea` are stored as XHTML:

```
<div data-classid="36f4349b-8093-492b-b616-05d8964e4c89"
     data-contentguid="4dd25c5f-66f0-41d0-9075-d0688638fb78"
     data-contentname="">{}</div>
<div data-classid="36f4349b-8093-492b-b616-05d8964e4c89"
     data-contentguid="dec4ca88-68b6-471b-a3ba-5398bb65ad68"
     data-contentname="" data-epi-content-display-option="narrow">{}</div>
<div data-classid="36f4349b-8093-492b-b616-05d8964e4c89"
     data-contentguid="eca36ce9-569c-4d8b-9d0a-a14255d89c25"
     data-contentname="" data-epi-content-display-option="narrow">{}</div>
<div data-classid="36f4349b-8093-492b-b616-05d8964e4c89"
     data-contentguid="d2ac8d27-be00-427a-8563-9a86cd062b42"
     data-contentname="" data-epi-content-display-option="narrow">{}</div>
```

Page links and references

To cache a page, store its ContentLink instead of the PageData instance since PageData is cached for 12 hours anyway.

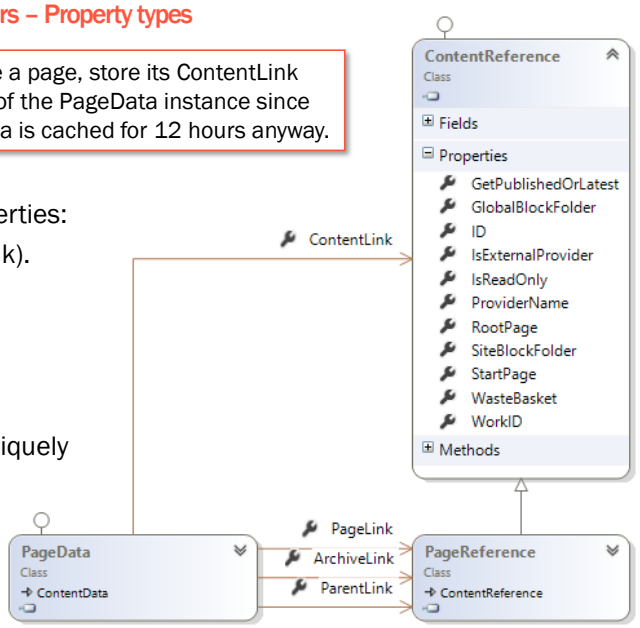
Every **PageData** instance has three "link" properties:

- **PageLink**: to itself (equivalent to ContentLink).
- **ArchiveLink**: to its archive when it expires.
- **ParentLink**: to its parent.

Each is an instance of a **PageReference**.

A **PageReference** is a **ContentReference** and a **ContentReference** has three properties that uniquely identifies a **ContentData** instance:

- **ID**: int
- **WorkID** (version)
- **ProviderName**



BackTypeResolver hard-coded resolutions

Property type in your custom content type	Resolved Type
System.Boolean (**bool**)	PropertyBoolean
System.Byte (**byte**), System.Int16 (**short**), System.Int32 (**int**), System.Int64 (**long**)	PropertyNumber
System.Decimal (**decimal**), System.Double (**double**), System.Single (**float**)	PropertyFloatNumber
System.**DateTime**	PropertyDate
System.**TimeSpan**	PropertyTimeSpan
System.String (**string**)	PropertyLongString
EPiServer.Core.**XhtmlString**	PropertyXhtmlString
EPiServer.DataAbstraction.**PageType**	PropertyPageType
EPiServer.**Url**	PropertyUrl
EPiServer.XForms.**XForm**	PropertyXForm

BackTypeResolver custom resolutions

If the type you used on your property is not in the table on the previous slide, then the **BackingTypeResolver** will scan all registered property definition types and choose the first where your type is equal to the type returned by **PropertyValueType**. This will make it possible to use the following types on properties in addition to the ones above.

Property type in your custom content type	Resolved Type
EPiServer.Core.**PageReference**	PropertyPageReference
EPiServer.Core.**ContentReference**	PropertyContentReference
EPiServer.Core.**CategoryList**	PropertyCategory
EPiServer.Core.**ContentArea**	PropertyContentArea
EPiServer.SpecializedProperties.**LinkItemCollection**	PropertyLinkCollection
YourNamespace.YourPropertyType	*YourRegisteredBackingPropertyType*

Collections of links

Use **LinkItemCollection** if you want to allow editors to *manually* control a collection of links to: pages, assets, external pages, e-mails. The links in a `LinkItemCollection` are stored as XHTML:

```
<links>
    <a href="~/link/80b857a10759444c8b2bf5c11f088b4b.aspx">Alloy Plan</a>
    <a href="~/link/a11b667f45cd4eafb05892243674b7c2.aspx">Alloy Track</a>
    <a href="~/link/6029af79956e4b82998820b3cd520b9f.aspx">Alloy Meet</a>
</links>
```

But it would cause a 404 if the page is removed or expires. An alternative would be to *automatically* generate the collection of links programmatically because this would allow the developer to apply filters that would remove any pages as soon as they are not published. For example, you could add a property that references a container page and then render the children of that page. Or you could write a search algorithm that returns a set of pages that match some criteria.

ContentReference class

ContentReference has static properties that point to some useful content instances:

- *PageReference*: **RootPage**, **StartPage**, **WasteBasket**
- *ContentReference*: **GlobalBlockFolder** (For All Sites), **SiteBlockFolder** (For This Site)

ContentReference has static methods:

- **IsNullOrEmpty(*ContentReference*)**
- **Parse(*string*)**: parses a string in the format ID[_WorkID[_ProviderName]] into a ContentReference.

An instance of ContentReference has some useful members:

- **ID**, **WorkID**, **ProviderName**: in combination, they uniquely identifies a version of content.
- **CompareToIgnoreWorkID(*ContentReference*)**: compares but ignores the WorkID; the IComparable.CompareTo method compares all the three properties above.
- **IsExternalProvider**: returns true if not stored in CMS. Use this instead of checking ProviderName.

PageReference class

PageReference has all the same members as ContentReference and some extras:

- **PageReference.ParseUrl(*string*)**: given a root-relative URL it will return a PageReference. If it cannot parse the string, it returns an empty reference. It does not throw an exception.
- **PageReference.IsValue(*PageReference*)**: returns true if it has a value, i.e. not null or empty. For example, StartPage is null before initialization, and empty if the start page hasn't been set.

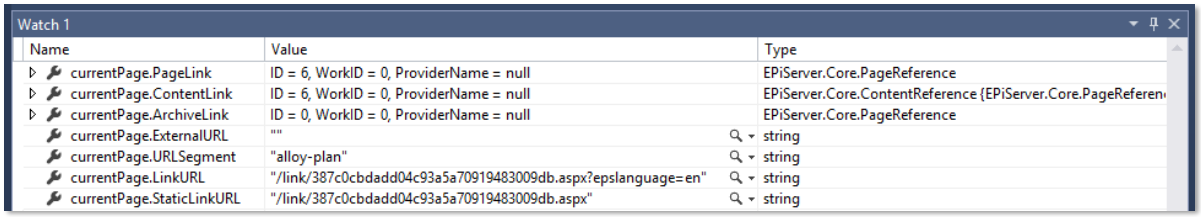
EPiServer.Core.ContentReferenceExtensions class defines an extension method to convert a ContentReference into a PageReference: **ToPageReference**().

Module D – Customizing Properties for Editors – Property types

Episerver links and URLs

What is the difference between the following properties of PageData? Note that some use the word "Link" and some use the acronym "URL" in their names:

- ArchiveLink, PageLink, ContentLink, ParentLink
- ExternalURL, LinkURL, StaticLinkURL, URLSegment



Module D – Customizing Properties for Editors – Property validation

How to custom validate a single property

```
public class OperaYearAttribute : ValidationAttribute
{
    public OperaYearAttribute()
    {
        ErrorMessage = "The first opera ever written was performed in 1597 in Florence in
Italy. It was called Dafne and the composer was Jacopo Peri.";
    }
    public override bool IsValid(object value)
    {
        return ((int)value) > 1597;
    }
}
```

```
[OperaYear]
public virtual int OperaWritten { get; set; }
```

How to custom validate multiple properties on a content type

```
public class EmployeePageValidator : IValidate<EmployeePage>
{
    public IEnumerable<ValidationError> Validate(EmployeePage instance)
    {
        var errors = new List<ValidationError>();
        if(instance.HireDate < instance.BirthDate) {
            errors.Add(new ValidationError {
                PropertyName = "HireDate",
                ErrorMessage = "An employee cannot be hired before they are born!",
                Severity = ValidationErrorSeverity.Warning,
                RelatedProperties = new string[] { "BirthDate" }
            });
        }
        return errors; // return an empty list if validation is okay
    }
```


IMetadataAware validation attributes

```
using EPiServer.Security;
using System.ComponentModel.DataAnnotations;
using System.Web.Mvc;
```

As well as inheriting from `ValidationAttribute`, you can implement the `IMetadataAware` interface to enable more powerful control.

```
public class PropertyEditRestrictionAttribute
 : ValidationAttribute, IMetadataAware
```

```
public string[] AllowedRoles { get; set; }
```

```
public PropertyEditRestrictionAttribute(
    string[] allowedRoles)
{
    AllowedRoles = allowedRoles;
}
```

```
public void OnMetadataCreated(ModelMetadata metadata)
{
    foreach (string role in AllowedRoles)
    {
        if (PrincipalInfo.CurrentPrincipal.IsInRole(role))
        {
            return;
        }
    }
    metadata.IsReadOnly = true;
}
```

Restricting access to who is allowed to edit a certain property

https://world.episerver.com/blogs/Linus-Ekstrom/Dates/2014/5/Restricting-access-to-who-is-allowed-to-edit-a-certain-property/

UIHint

Decorate the property with UIHint to control how the property is edited.

A string property is edited in a textbox as default:

```
public virtual string MyProperty { get; set; }
```

If we need a multiline text area instead, we can decorate it with UIHint.Textarea or UIHint.LongString:

```
[UIHint(UIHint.Textarea)]
public virtual string MyProperty { get; set; }
```

You can create custom UIHints when needed:

```
public static class SiteUIHints
{
    public const string Contact = "contact";
    public const string Strings = "StringList";
}
```


EditorDescriptor

- Use EditorDescriptor **to register a custom editor**
- Decorate it with UIHint to be able to use it from properties

```
namespace EPiServer.Templates.Alloy.Business.EditorDescriptors
{
 [EditorDescriptorRegistration(
        TargetType = typeof(PageReference),
        UIHint = "MyHint")]
    public class MyCustomEditor : EditorDescriptor
    {
```

Best practice tip:
Create static class with string constants to provide UIHint values

UiWrapperType

You can define the kind of editor to use when the user is in ”on-page-edit” view.

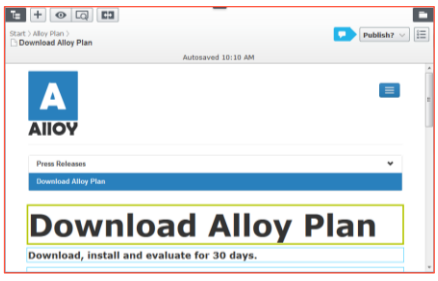
This is done in the custom EditorDescriptor class.

You override the method ModifyMetadata on EditorDescriptor and add the custom editor settings as in the example below:

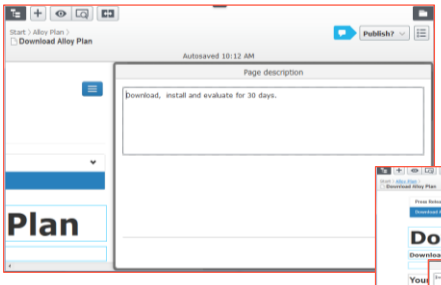
```
EPiServer.Shell.UiWrapperType

//Use in a custom EditorDescriptor:
metadata.CustomEditorSettings["uiWrapperType"] = UiWrapperType.Flyout;
```


UiWrapperType

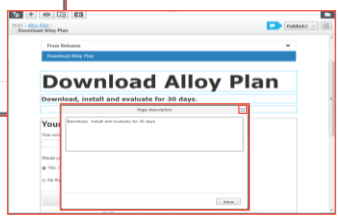
`UiWrapperType.ContentEditable`



`UiWrapperType.Flyout`



`UiWrapperType.Floating`



Overview of Dojo

- Episerver UI uses Dojo Toolkit as the JavaScript framework.
- Dojo is an open source framework.
- One objective is that developers should not need to handle Dojo in most Episerver web projects, however, when needed it is a powerful tool to use.
- Components:
  - **Dojo**: Core API of the framework. DOM manipulation, class declaration, event listening, messages and asynchronous requests
  - **Dijit**: User interface system built on top of the Dojo core. Widget system used to handle visual elements in a modular manner.
  - **Dojox**: Sub-projects built on top of the Dojo core. Dojo plugins and new features.

Dojo – the minimum

```
[Component(
    PlugInAreas = "/episerver/cms/assets",
    Categories = "cms",
    WidgetType = "alloy.components.CustomComponent",
```

- Register plugin
  - decorate with WidgetType, mapped to .js file
- Create .js file
  - define
    dojo/_base/declare – is used so the “declare” method can be called
    dijit/_WidgetBase – is required since it is a widget
  - Add function
    function (declare, _WidgetBase)
  - Return of a widget class with one method to create a debug message to the console. Since "declare" doesn’t have a callable constructor it should not be part of the constructor arguments to the class.

Dojo – the minimum

The full .js code:

```
define([
    "dojo/_base/declare",
    "dijit/_WidgetBase"
], function (
    declare,
    _WidgetBase) {
    return declare("alloy.components.CustomComponent", [_WidgetBase], {
            postCreate: function () {
                console.debug("Method postCreate is called");
                this.inherited(arguments);
            }
    });
});
```


Dojo – Knowledge

- Learn more about Dojo
  - http://dojotoolkit.org
  - Episerver SDK Developer Guide

- References:
- http://world.episerver.com/Blogs/Linus-Ekstrom/Dates/2012/7/Creating-a-custom-editor-for-a-property/
- http://episerver.se/developing/plugins/dojo-widgets/simple-dojo-widget
- Also see the StringList editor in the Alloy sample site.

Module D – Customizing Properties for Editors – On-Page Editing

Taking control of client-side rendering

CMS UI 10.12 or later includes options to better support the On-Page Editing (OPE) experience for websites that want to handle the view on the client-side with JavaScript frameworks such as Angular.

1. To stop the CMS UI from replacing the DOM when an editor changes the value of a property, add the HTML attribute data-epi-property-render="none"
2. Whenever a save happens we will publish the details on a topic called "beta/contentSaved"

Taking control of client-side rendering in OPE

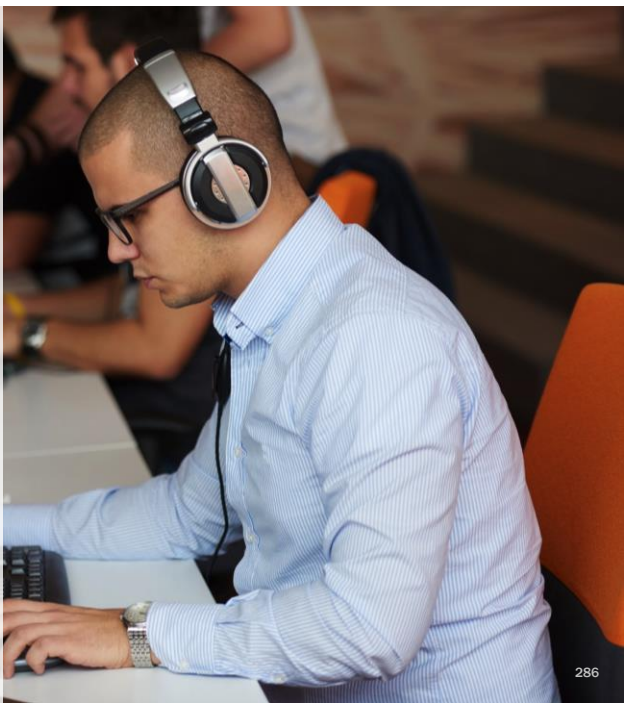
https://world.episerver.com/blogs/john-philip-johansson/dates/2017/10/taking-control-of-client-side-rendering-in-ope-beta/

https://world.episerver.com/blogs/john-philip-johansson/dates/2017/12/taking-more-control-of-client-side-rendering-in-ope-beta2/

Module D – Customizing Properties for Editors

Exercises for Module D

1. Applying CSS to a property editor.
2. Selecting choices for property values using SelectionFactories.
3. Using UIHints to select an editor.
4. Customize any property at runtime using an EditorDescriptor.
5. Create a custom editing experience for date-only pickers using Dojo.



Episerver CMS Advanced Development

Module E
Rendering, Personalizing, and Indexing Content

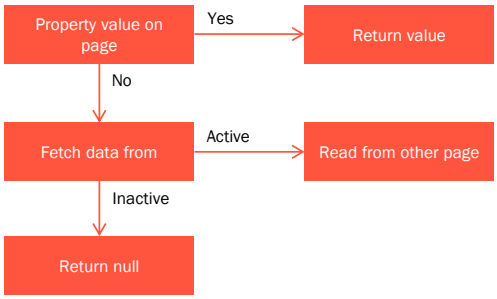
When building a site today you need to consider the different channels that the content can be presented in. Content are also often re-used in several places and need to be displayed differently depending on the context. And you need to index content to enable your visitors to easily search for it.

Topics

- Property rendering
- Rendering and resolving templates
- Display channels
- Personalizing with visitor groups
- Personalizing with smart content
- Custom rendering
- Indexing content with Episerver Search

Module E – Rendering, Personalizing, and Indexing Content – Property rendering

Access a property



- Content delivered from API is in read only mode, for example:
  `Object propertyValue = CurrentPage["PropertyName"];`
- If the page is write enabled the property accessor will only return properties that are defined on the page.

Module E – Rendering, Personalizing, and Indexing Content – Property rendering

What does @Html.RenderEPiServerQuickNavigator() do?

```
▼<ul id="epi-quickNavigator">
  ▼<li class="epi-quickNavigator-editLink">
    ▼<a href="/EPiServer/CMS/?language=en#cont
       <span>EPiServer</span>
     </a>
   </li>
  ▼<li class="epi-quickNavigator-dropdown">
     <a id="epi-quickNavigator-clickHandler" c
     "javascript:void(0)"></a>
    ▼<ul id="epi-quickNavigator-menu" style="d
      ▼<li>
         <a href="/EPiServer" target="_self">Dashboard</a>
       </li>
      ▼<li>
         <a href="/EPiServer/CMS/?language=en#context=epi.cms.contentdata:///5" target=
         "_self">CMS Edit</a>
       </li>
     </ul>
   </li>
 </ul>
```

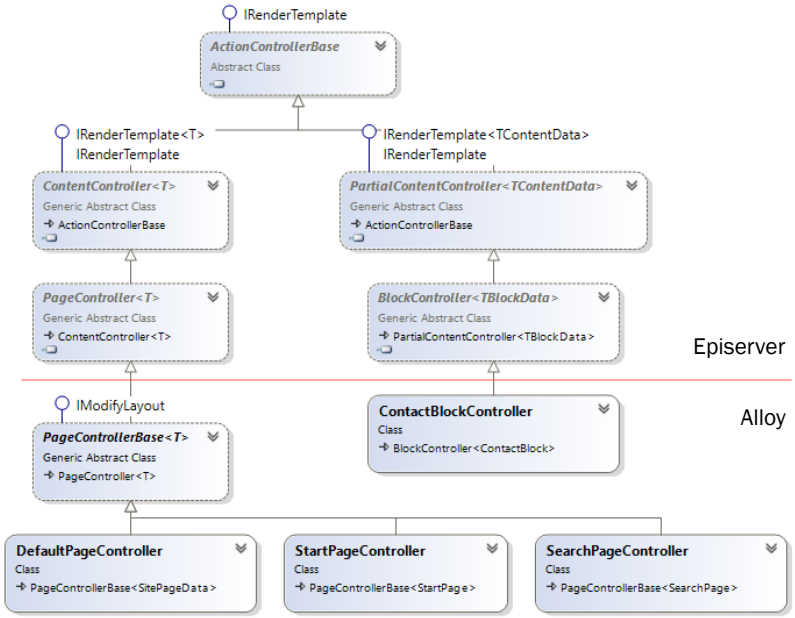
```
▼<body>
   <link rel="stylesheet" type="text/css" href="/Util/styles/quicknavigator.css">
   <script type="text/javascript" src="/Util/javascript/quicknavigator.js"></script>
  ▼<script type="text/javascript">
                //<![CDATA[
                (function () { new epi.QuickNavigator({"menuItems":{"dashboard":
     {"caption":"Dashboard","url":"/EPiServer","javascript":null,"enabledScript":"true","im
     ageUrl":null},"editMode":{"caption":"CMS Edit","url":"/EPiServer/CMS/?
     language=en#context=epi.cms.contentdata:///5","javascript":null,"enabledScript":"true"
     ,"imageUrl":null}},"menuTitle":"EPiServer","defaultUrl":"/EPiServer/CMS/?
     language=en#context=epi.cms.contentdata:///5"}); }());
                //]]>
   </script>
```

Logged in visitors will see the orange ***epi*** logo and be able to use it to quickly edit the current page, or go to their Dashboard.

Rendering templates

Episerver CMS's template resolver looks for any class that implements **IRenderTemplate<T>** where **T** is the PageData-derived type that represents the requested page instance.

In practice, you will usually define a class that derives from **PageController<T>** , **PartialContentController<T>**, or **BlockController<T>** as in this example hierarchy in Alloy.



Episerver

Full and partial page templates

If you have a page type named **EmployeePage** with properties for:

- EmployeeCode, FirstName, LastName, Department, BirthDate, HireDate, FireDate.

You could have at least two page templates for it:

- `EmployeePageController : PageController<EmployeePage>`
  when a normal, full-page request is made for the page.
  - `~/Views/EmployeePage/Index.cshtml`:
    the view would typically output ALL the page's properties and have a layout.
- `EmployeePartialPageController : PartialContentController<EmployeePage>`
  when the page is dropped into a ContentArea.
  - `~/Views/EmployeePartialPage/Index.cshtml`:
    this view would typically output a subset of the page's properties WITHOUT a layout.

Episerver

294

Partial rendering

- Using partial rendering of pages for the “teasers”
  - Drop a page into a content area or a XhtmlString
  - Partial rendering
  - Content fetched from the page properties instead of separate teaser block

```
public partial class MyPageTeaser : PartialContentController<MyPage>
{ }
```

Or, without controller:

```
public void Register(TemplateModelCollection viewTemplateModelRegistrator)
{
viewTemplateModelRegistrator.Add(typeof(MyPage), new TemplateModel
    {
        Name = "PagePartial",
        Inherited = true,
        AvailableWithoutTag = true,
        Path = "~/Views/Shared/PagePartials/Page.cshtml"
    });
}
```


Page partial or block?

When is it motivated to have a partial renderer for a page and drop it into a content area and when should the content be in a block instead? Guidelines in brief:

- Editorial content and content that you want to be able to bookmark the URL to should be in pages.
- Functions (for example “share this on facebook”, “basic news list” or “contact us form”) could/should be in blocks.
- All page types that you want to be able to promote by drag-dropping them into a content area should have a good “neat” partial renderer*. If a page is promoted by using a block, for example a special campaign using a teaser, this will create an overhead for the editor since they need to manage the block as a separate item in parallel with the page (create the block instance, link it to the page, add content and publish it). If a teaser block is not needed for a page – don’t use one just for the sake of it, use a partial renderer instead.

Module E – Rendering, Personalizing, and Indexing Content – Rendering and resolving templates

Page template resolver

Display Channels are "tags" that are set automatically by code logic. Multiple display channels can be active simultaneously.
Display Options are "tags" that are set manually by an editor. Only one display option can be set at once.

Request	Display Channel	Display Option	Controller	Template Descriptor attribute applied to controller
Full page e.g. GET /.../.../			Page Controller<T>	
	Mobile			[TemplateDescriptor(Tags = new string[] { "mobile" })]
	Print			[TemplateDescriptor(Tags = new string[] { "print" })]
Partial page e.g. drag and drop page into ContentArea			PartialContent Controller<T>	
	Mobile			[TemplateDescriptor(Tags = new string[] { "mobile" })]
	Print			[TemplateDescriptor(Tags = new string[] { "print" })]
		Narrow		[TemplateDescriptor(Tags = new string[] { "narrow" })]
		Wide		[TemplateDescriptor(Tags = new string[] { "wide" })]
		Full		[TemplateDescriptor(Tags = new string[] { "full" })]

Episerver 297

Module E – Rendering, Personalizing, and Indexing Content – Rendering and resolving templates

TemplateResolver example (MVC)

Controllers/registered templates

```
[TemplateDescriptor(Inherited = true)]
public class DefaultPageController : PageControllerBase<PageData>
```

```
public class StandardPageController : PageControllerBase<StandardPage>
```

```
viewTemplateModelRegistrator.Add(typeof(PageData), new TemplateModel
    {
        Name = "PagePartial",
        Inherit = true,
        AvailableWithoutTag = true,
        Path = PagePartialPath("Page.cshtml")
    });
```

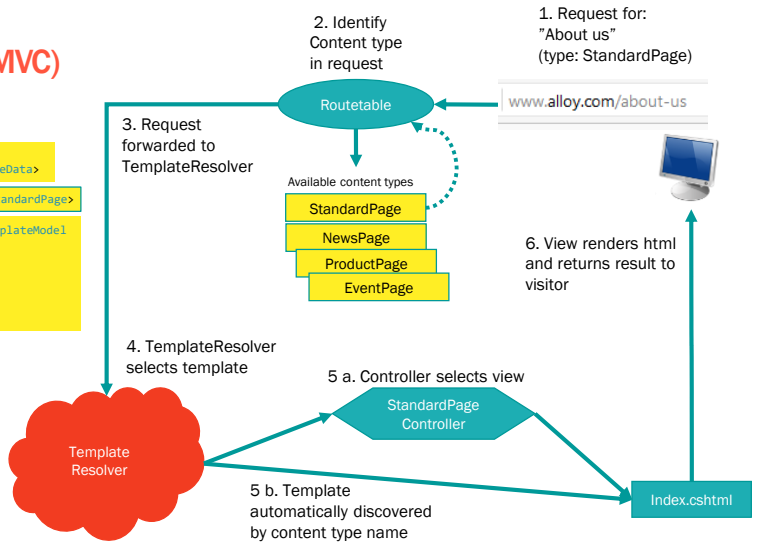
Views

```
/StandardPage/Index.cshtml
@model StandardPage
```

```
/NewsPage/Index.cshtml
@model PageViewModel<NewsPage>
```

```
/Shared/PagePartial.cshtml
@model PageData
```

```
/Shared/EventBlock.cshtml
@model EventBlock
```



Episerver 298

TemplateResolver – algorithm

TemplateResolver automatically selects which template to use depending on current context:

1. **Rendering mode** – page or partial rendering.
2. **Tags** – Any tag in the request (tag[] = [“SideBar”]).
3. **DisplayChannel** – Which channel is active?
4. Use closest, default and **not inherited template**.
5. Use closest, **default template**.
6. Use **closest template**. With “closest” above means the template model with shortest “inheritance chain”. That means that a template that is registered direct for the model will be preferred before a template registered for a base class. It is possible to register templates for interfaces as well.

TemplateResolver – events

It is possible to listen to raised events to control or override which template to use:

- TemplateResolver.**TemplateResolving** is raised before the selection chain is started.
- TemplateResolver.**TemplateResolved** is raised after the selection chain is completed.

Why would you want to override the template resolver either before or after it has been actioned?

In an Enterprise site, where you can have multiple start pages, same codebase, same page types, but want to have different rendering for _some_ of the page types.

Example: The Start page template is used for 3 of 4 sites, but for one site you need a completely different rendering. It is the same page type though. You can tailor the selector to do this with some custom code, listening to the TemplateResolver event(s).

Sample code: A code example that demonstrates how to exchange the template for mobile requests is available on the TemplateResolver class in the downloadable EpiserverCMS SDK (look at EPiServer.Web.TemplateResolver).

Several templates for one content

Full-page template variations using the same model:

- web channel
- mobile channel
- PDF channel

Partial template variations using the same model:

- ContentArea tagged “A”
- ContentArea tagged “B”
- DisplayOption using tag “C”

Single template, multiple renderings

- The following example shows how to set CSS class depending on the active channel

```
bool mobileDisplayChannelActive =
        ServiceLocator.Current.GetInstance<DisplayChannelService>()
            .GetActiveChannels(this.ControllerContext.HttpContext)
            .Any(c =>String.Equals(c.ChannelName,"mobile",StringComparison.OrdinalIgnoreCase));

if (mobileDisplayChannelActive == true)
{
    string cssClass = "Mobile";
}
```


Overview of Display Channels

- Used to control rendering depending on the request, for example
  - Different browsers
  - Mobile visitors
- Different templates
  - Use different templates for different channels
- Single template
  - All channels use the same template, but it can render content suitable for different devices

Why do we need Display Channels? Responsive vs. adaptive design

Responsive design

- Happens on the client-side using HTML5, CSS3, and JavaScript
- Same response for all requests

Adaptive design

- Happens on the server-side using display channels
- Customize response for each request…
  - …and therefore can minimizes size of response so better for low bandwidth customers.

Many sites use both.

How to create a display channel

```
public class MobileDisplayChannel : DisplayChannel
{
    public override string ChannelName
   {
           get { return RenderingTags.Mobile; }
   }
   public override bool IsActive(HttpContextBase context)
   {
           return context.Request.Browser.IsMobileDevice;
   }
}
```

Add MobileDisplayChannel.cs in folder Business/Channels

Inherit from EPiServer.Web.DisplayChannel → Registered during initialization

IsActive
Add logic that decides if channel is considered active

TemplateDescriptor

Use tags to make a template selected for a channel

- Two templates registered for **MyBlock**
- Template with tag matching ChannelName of active channel is selected

```
namespace EPiServer.Framework.Web
{
    public static class RenderingTags
    {
        public const string Preview = "Preview";
        public const string Edit = "Edit";
        public const string Header = "Header";
        public const string Footer = "Footer";
        public const string Article = "Article";
        public const string Sidebar = "Sidebar";
        public const string Mobile = "Mobile";
        public const string Empty = "Empty";
    }
}
```

```
public partial class MyBlockControl : BlockControlBase<MyBlock>
{ }

[TemplateDescriptor(Tags = new string[] { RenderingTags.Mobile })]
public partial class MyBlockMobileControl : BlockControlBase<MyBlock>
{ }
```

Module E – Rendering, Personalizing, and Indexing Content – Display channels

DisplayChannelService

- EPiServer.Web.DisplayChannelService
- Retrieved via ServiceLocator
- Determines active channel

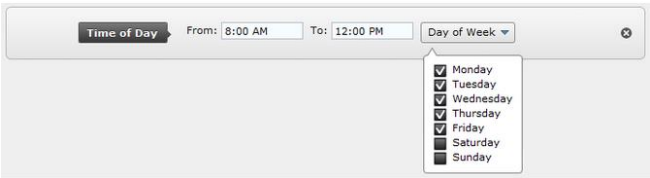
```
List<DisplayChannel> channels =
    ServiceLocator.Current.GetInstance<DisplayChannelService>()
    .GetActiveChannels(new HttpContextWrapper(HttpContext.Current))
    .ToList();
```

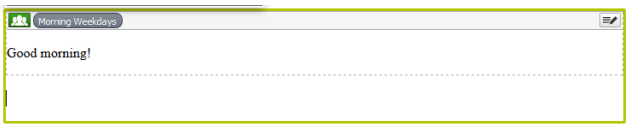
Episerver 308

Module E – Rendering, Personalizing, and Indexing Content – Personalizing with visitor groups

Personalization

Personalization provides the possibility to personalize website content to specific visitor groups so that different content will be displayed to different visitors, bringing a more personalized website experience. The personalization feature is based on the visitor group criteria concept. All Episerver products contain criteria that are ready to use, but it is also possible to develop your own criteria. This document explains the concept and describes the procedure for creating customized visitor group criteria.



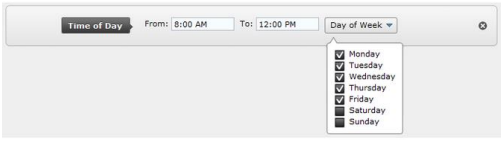


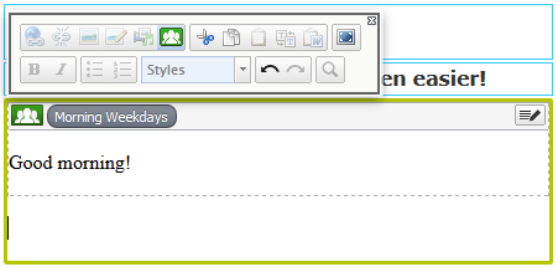
Episerver 310

The visitor group criterion

From a development standpoint, a visitor group criterion is a combination of (at least) two classes:

1. A model class that stores and persists user input from the UI.
2. A criterion class that evaluates the context and the data stored in the model to determine if the criteria is fulfilled or not.





Creating a model class

The model class stores and persists user input from the UI and provides the criterion class with easy access to the settings. The model class must implement ICriterionModel and, since instances of the model class will be persisted to the Dynamic Data Store, IDynamicData. The best way of implementing those interfaces is by inheriting from CriterionModelBase which contains standard implementations.

The only method you must override is CriterionModelBase.Copy, if you are working with simple value type properties it is sufficient to let your override call base.ShallowCopy. If your model has reference type properties and you want to make sure that each instance of the model has its own copy of the referenced objects you need to do extend the Copy override with more logic

Creating a model class – code example

- Model class + Criterion class

```
using EPiServer.Personalization.VisitorGroups;

public class MyCustomCriterionModel : CriterionModelBase
{
    [Required]
    public string TimeFrom { get; set; }

    [Required]
    public string TimeTo { get; set; }

    [DojoWidget(SelectionFactoryType = typeof(WeekdaySelectionFactory))]
    public WeekdayList DayOfWeek { get; set; }

    public override ICriterionModel Copy() { return base.ShallowCopy(); }
}
```


Creating a criterion class

The criterion class should evaluate the HTTP context and the data stored in the model to determine if the criteria is fulfilled or not. The connection between the criterion and model classes is created via CriterionBase – the base class that must be used for the criterion class – which is a generic class that accepts ICriterionModel parameters. The only method you must override is CriterionBase.IsMatch which is the central method for a criterion, it is the method that will be called when evaluating if a user is a member of a visitor group.

The criterion class must also be decorated with VisitorGroupCriterion attribute, which identifies your class as a criterion and makes it available for use.

- **Category**: The name of the group in the criteria picker UI where this criterion will be located. Criteria with the same Category value will be grouped together.
- **DisplayName**: A short name that is used to identify the criterion in menus and visitor groups.

Creating a criterion class – code example

- Model class + Criterion class

```
using EPiServer.Personalization.VisitorGroups;

[VisitorGroupCriterion(Category = "URL Criteria", DisplayName = "Custom Criterion")]
public class MyCustomCriterion : CriterionBase<MyCustomCriterionModel>
{
    public override bool IsMatch(System.Security.Principal.IPrincipal principal,
                                 HttpContextBase httpContext)
    {
        //Evaluation code here. Example: if visitor timezone is within input range
    }
}
```


Benefits of Episerver Advance's smart content

Previously, you would have to rely on manual content selection to create home pages, article listings, or landing pages. Episerver Advance, our smart content solution, does the heavy lifting for you. Thanks to tagging and Episerver's search relevance engine, Episerver Find, content is automatically selected for each visitor based on visitor profile, interest, or work role.

You will learn more in *Module G: Indexing Content using Search and Find*.

For content marketing: Highly relevant content suggestions are presented on the first visit, and through repeat visits.

For articles and news: Keep visitors engaged and on the site with content suggestions that are relevant to topics, and optimized for popularity and newness.

For intranets and portals: Present information and documents that are relevant to each employee, enhancing content discovery and reducing time spent on information search.

Benefits of Episerver Advance's smart content

Instead of time-consuming and error-prone rules-based personalization, Episerver Advance makes use of autonomous personalization – where machine learning based algorithms are used to inject the right content at the right time for every single visitor.

Personalized content on first page view: Episerver uses contextual data, such as ad clicks, geolocation, and organization affiliation to present relevant content on start pages, content listings and landing pages.

Automatically surfaced information: For many websites and portals, it is deep content that is most likely to be relevant to a visitor. Episerver uses tagging and content filtering to surface content that is more likely to serve the visitor's needs.

Interactive content drill-down: From the first content selection, visitors are quickly able to drill down into large content repositories thanks to Episerver's intuitive guided navigation.

Customizing the rendering of ContentAreas

A content area is used to display content by drag'n'drop.

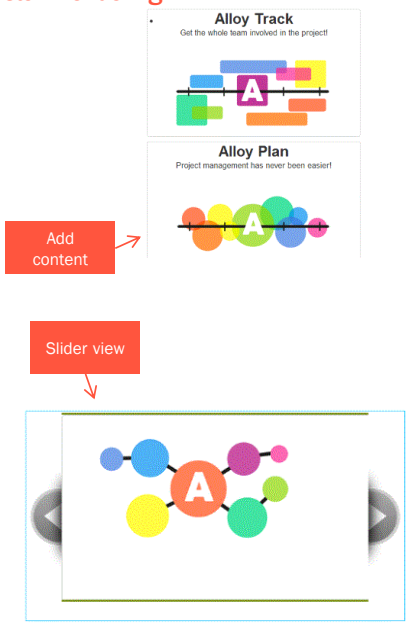
We are used to customizing the blocks and pages that are displayed by the content area.

It is also possible to customize the content area itself

For example:

- Add several content items to one content area
- The content area will display a gallery slider showing one items at a time

http://world.episerver.com/Blogs/pezi/Dates/2013/5/Create-an-animating-slider-with-content-area/



About Episerver Search

- Episerver Search is a simple but effective solution that will cover the needs of any basic search.
- Built on the Lucene indexer.
- Deployed through NuGet packages
  - Indexing service: `EPiServer.Search`
  - Episerver CMS integration : `EPiServer.Search.Cms`

Built-in features of Episerver Search

- Full-text search
- Global search
- Static facets
- Event driven indexing for instant search results
- Index any type of content
- Access rights-based search result filtering
- Pluggable search interface with `ISearchProvider`

If more advanced features are needed, then use Episerver Find.

Configuring Episerver Search

```
<episerver.search active="true">
  <namedIndexingServices defaultService="serviceName">
     <services>
       <add name="serviceName"
            baseUri="http://.../IndexingService/IndexingService.svc" accessKey="local" />
     </services>
  </namedIndexingServices>
  <searchResultFilter defaultInclude="true">
     <providers />
  </searchResultFilter>
</episerver.search>
```



```
using EPiServer.Search.Queries.Lucene;
```

Searching indexed content with Episerver Search

Type	Method	Parameter(s)	Return Type
SearchHandler	GetSearchResults	IQueryExpression e.g. FieldQuery, GroupQuery, and so on.	SearchResults

```
private readonly SearchHandler searcher;
```

```
var query = new FieldQuery("alloy");
SearchResults results =
    searcher.GetSearchResults(query, page: 1, pageSize: 10);
int hits = results.TotalHits;
Collection<IndexResponseItem> pageOfItems = results.IndexResponseItems;
```

Advanced options

- Use **GroupQuery** to create AND, OR & NOT groupings
- Limit to certain Language Branches
- Limit to certain Content Types (MediaData, for example)
- Add root pages to your search to limit results to pages below that page
- Search based on access rights

```
var pageTypeQuery = new GroupQuery(LuceneOperator.AND);
pageTypeQuery.QueryExpressions.Add(new ContentQuery<PageData>());
pageTypeQuery.QueryExpressions.Add(new FieldQuery(languageBranch, Field.Culture));
```

```
var accessRightsQuery = new AccessControlListQuery();
accessRightsQuery.AddAclForUser(PrincipalInfo.Current, context);
query.QueryExpressions.Add(accessRightsQuery);
```


Limitations and load balancing with Episerver Search

Limitations

- Search will not index blocks in content areas (by default).
  Episerver CMS Advanced Development course shows how to implement this with code.

Resetting the index

- To manually reset the Episerver Search index, enter the following URL:
- `/EPiServer/EPiServer.Search.Cms/IndexContent.aspx`

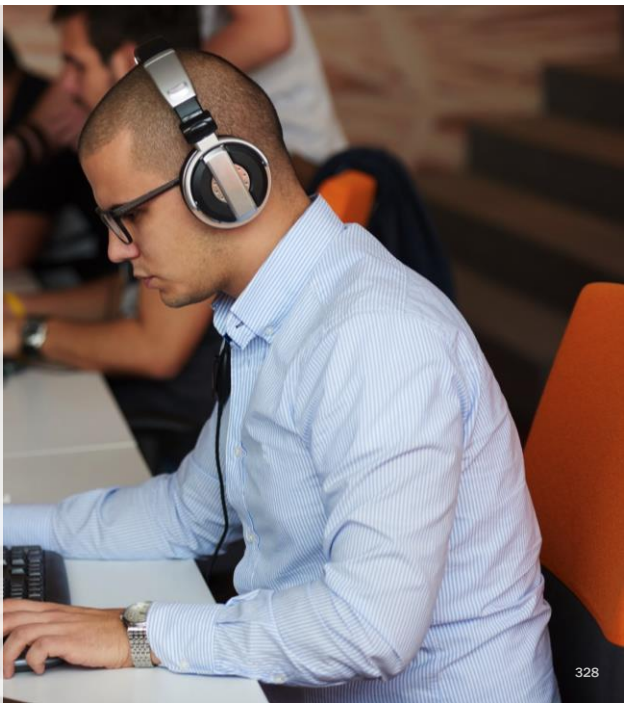
Load Balancing

- In a load balanced environment, the supported scenario is to install the search service on one of the machines, and configure that machine as the search service for other machines.

Module E – Rendering, Personalizing, and Indexing Content

Exercises for Module E

1. Using UIHints to apply display templates
2. Creating a PDF display channel
3. Detecting visitor groups with cookies
4. Adding fields to Episerver Search



Episerver

328

Episerver CMS Advanced Development

Module F
Extending with Plug-ins and Add-ons

With Episerver extensions to your site can be installed via NuGet. This can be anything from a new content type or visitor group criterion to installing a new version of the UI. As a developer you need to know what add-ons are and the options available to package your custom modules.

Episerver

329

Module F – Extending with Plug-ins and Add-ons

Topics

- Overview
- Extension points
- Developing plug-ins and gadgets
- Developing add-ons
- Example add-ons

Module F – Extending with Plug-ins and Add-ons – Overview

Why extend Episerver CMS?

For visitors to the website:

- Develop templates providing the desired web design and functionality to make this possible.

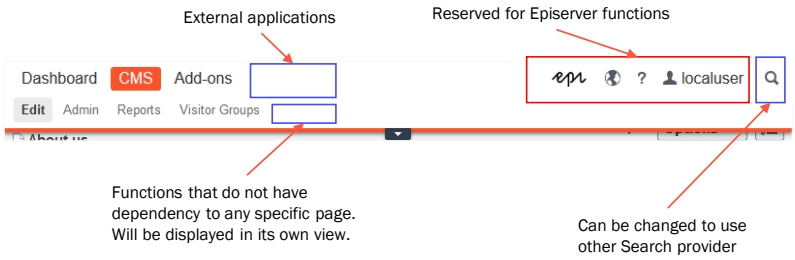
For users of the Episerver UI (Editors, Administrators):

- Build a content structure, define content types and properties supporting the specific needs.
- Enhance Episerver through extension mechanisms such as plug-ins and custom property types.
- Examples of using plug-ins to make services that helps editors:
  - Blog posts automatically created in the correct place in the page tree according to month and year (creates the folders if they don't exist)
  - Custom property that lets the editor select longitude and latitude for a location to use with for instance Google Maps.

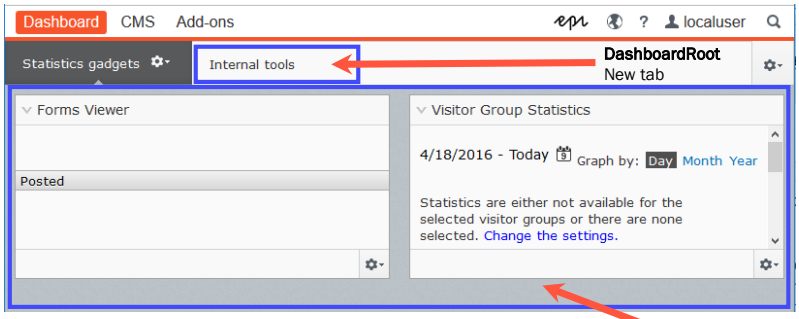
Terms for extension points

- **Plug-in**: a visual extension for administrators or editors that is visible to all, e.g. scheduled job, custom tool, reports, and so on.
- **Gadget**: a visual extension for editors to choose to add to their Dashboard, Navigation pane, or Assets pane. Each user has their own configuration of gadgets.
- **Add-on**: a way to package an extension for distribution via Episerver's NuGet feed. Plug-ins and gadgets do not need to be packaged as an add-on if you are deploying internally.

Extending the Global menu with plug-ins

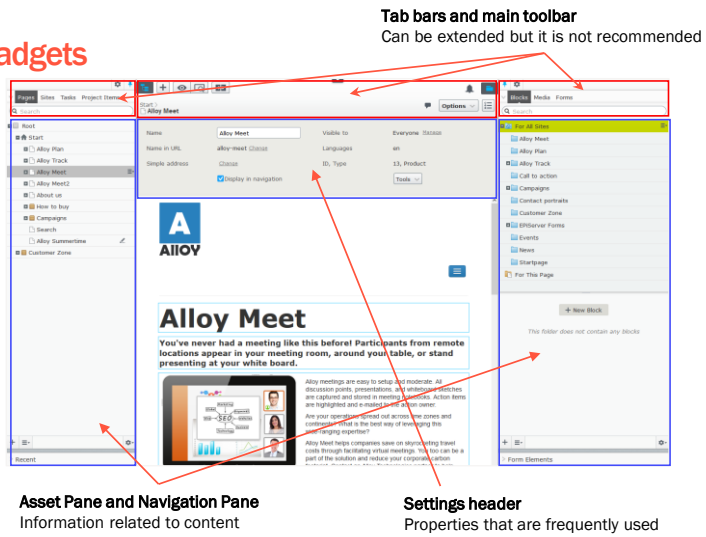


Extending the Dashboard with gadgets



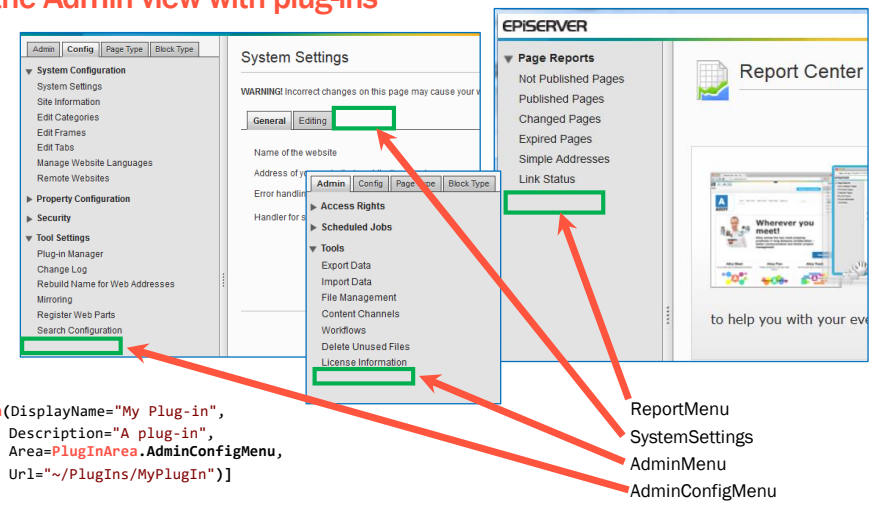

```
[Component(Title="My Dashboard plug-in",
           Description="A plug-in",
           PlugInAreas=PlugInArea.DashboardDefaultTab,
           Categories="dashboard")]
```


Extending the Edit view with gadgets



Module F – Extending with Plug-ins and Add-ons – Extension points

Extending the Admin view with plug-ins




```
[GuiPlugIn(DisplayName="My Plug-in",
           Description="A plug-in",
           Area=PlugInArea.AdminConfigMenu,
           Url="~/PlugIns/MyPlugIn")]
```


Module F – Extending with Plug-ins and Add-ons – Developing plug-ins and gadgets

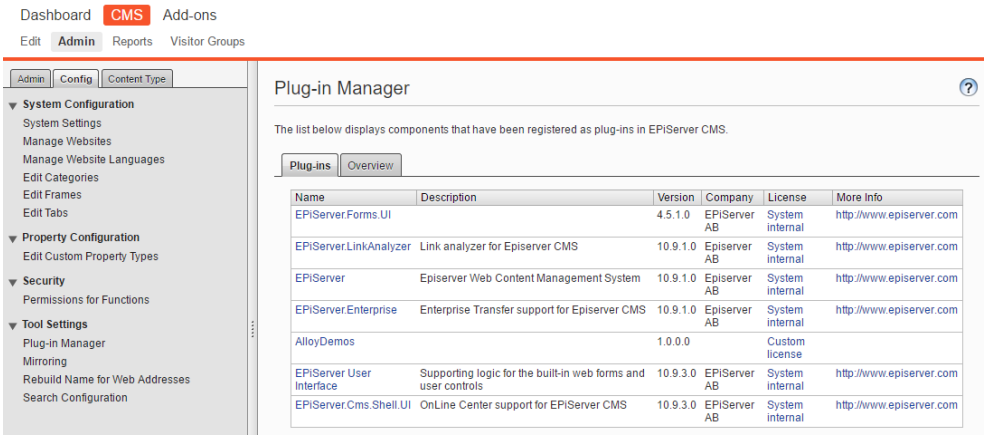
Extending using Episerver plug-ins

- Episerver plug-ins offer a flexible way to extend Episerver
- Deploy them as part of a solution, or stand-alone
  - Automatic detection and inclusion in the web site (from /bin and /modulesbin directories)
- Episerver CMS uses an automatic registration architecture for handling extension points
  - All plug-ins that inherit or uses the classes **Component** or **PlugInAttribute** are loaded at startup
- Plug-in related classes, constants and interfaces for GUI components are documented in the EPiServer.Shell.ViewComposition namespace in the Episerver Framework SDK.
- Plug-in related classes, enumerations and interfaces for Admin are found documented in the EPiServer.PlugIn namespace in the Episerver CMS SDK.
- See the inheritance hierarchy for PlugInAttribute class for a complete list.

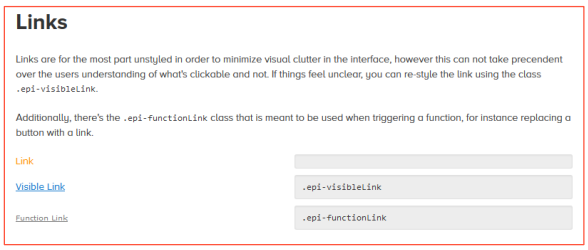
Plug-in manager

Navigate to **CMS | Admin | Config | Plug-in Manager**

Shows Episerver CMS version e.g. 10.9.1.0, and other shell modules deployed as plug-ins.



Episerver front-end style guide: http:/ux.episerver.com



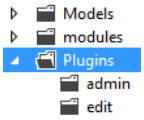
The style guide is a living document meant to assist both Episerver and external developers explore our theme and get an overview of what classes and styles are available.

When creating your own plug-ins

- Separate the edit and admin parts
- Remove UI-plug-ins from the public facing server
- Set access rights on the location paths in config, to ensure that they cannot be reached by unauthorized users accessing the page directly

References and examples:

- http://world.episerver.com/Blogs/Mari-Jorgensen/Dates/2010/11/Protect-your-plugins/
- http://world.episerver.com/FAQ/Items/Securing-plug-in-files/




```
<!-- Restrict access to files beneath the Plugins folder -->
<location path="Plugins/admin">
  <system.web>
    <authorization>
      <allow roles="WebAdmins, Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
<location path="Plugins/edit">
  <system.web>
    <authorization>
      <allow roles="WebAdmins, WebEditors, Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
```


What are add-ons? What is the shell? What are modules?

Developing shell modules is the way you integrate your add-on into the Episerver platform. Episerver add-ons are shell modules.

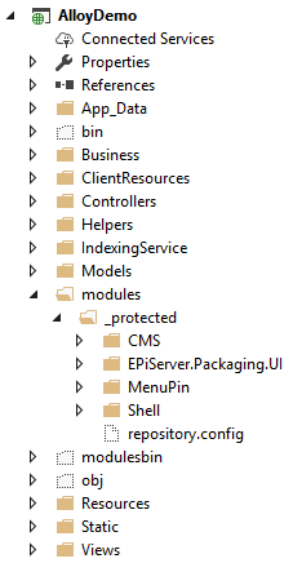
Episerver CMS is shipped with three shell modules:

- **Shell**: dashboard and resources shared between multiple products
- **CMS**: shell resources specific to the CMS
- **EPiServer.Packaging.UI**: interface files for the Add-on store

If you were to install an add-on such as Episerver Forms, you would see additional modules have been deployed:

- **EPiServer.Forms**: shell resources specific to the Forms add-on

How to package add-ons video (90 minutes):
http://fast.wistia.net/embed/iframe/4dhm5342lt?videoFoam=true



Add-on levels

Three levels that developers can participate in:

- Developer Add-on
- Site Owner Add-on
- Verified Solution Add-on

Site Owner Add-Ons and Verified Solution Add-ons are formally part of the add-on program and require pre-approval from Episerver prior to their creation/integration.

Developer add-ons

- Support when developing sites
- Open source
- Provided as NuGet packages
- Requirements is that it is open-source, created by an Episerver-certified developer.
- Examples:
  - Blob Converter
  - PowerSlice
  - YouTube Block
  - Geta.tags

https://github.com/episerver/PowerSlice

The Developer Add-On program allows all and any Episerver-certified developer to create open-source Add-Ons and make it available to the Episerver developer community.
Developer Add-ons are not verified by Episerver, and not supported by Episerver.
The developer submits the add-on on nuget.episerver.com, where is it reviewed by Episerver and generally approved if coming from a known and trustworthy source. Once approved it is available in the developer NuGet feed.

Site Owner add-ons

- Enhance or enable new functionality
- Tested by Episerver
- One-click installs from the Add-On Store

- Examples:
  - SiteAttention
  - Mogul SEO
  - Translations.com

Site Owner Add-Ons tend to be smaller installs and more "software" versus "solution".
To enter as a Site-Owner Add-On, the partner/developer/application must pass through an application and approval process. There is a high level of testing that is done, and co-sharing of marketing, sales, support and training information between us, which will enable greater understanding of your Add-On by our sales representatives worldwide.

Verified Solution add-ons

- Products or services
- Tested by Episerver
- http://www.episerver.com/AddOns/
- License fee might apply

- Examples:
  - ImageVault
  - Silverpop
  - Agility Multichannel
  - Celum
  - Perfion

Verified Solution Add-Ons are the most flexible type of partnership. They enable key strategic capabilities, and are often larger platform solutions where the integration is not a "one-click install" integration - either the vastness or complexity of the platform prevents a simplistic integration. Verified Solution Add-Ons are tested in the same manner as Site-Owner Add-Ons, in addition they are tested for basic functionality and use cases, which enables that partner to post marketing information of that Add-On via Episerver.com and have their co-branded marketing and sales information passed to all Episerver sales representatives globally.

Module F – Extending with Plug-ins and Add-ons – Developing add-ons

Developing add-ons

- Add-ons are packages
  - that extend and improve independently from product releases
- For example
  - Plug-ins, scheduled jobs, gadgets, editing UI components, data stores, typed pages, blocks, templates
- Add-ons are deployed as Shell modules
  - Virtual path to views and client resources
- Delivered as a NuGet package
  - containing all add-on resources and assemblies
- Installation
  - Installed from the site's web interface, no direct access needed to the server machine. (*)

Module F – Extending with Plug-ins and Add-ons – Example add-ons

Supports DXC Service	Yes
Requires license	No

Google Analytics for Episerver

https://world.episerver.com/add-ons/google-analytics-for-episerver/

- Fully integrated, adding insight and context to their content creation process.
- Constantly improve the user journey and customer experience on any type of web, e-commerce, mobile or social site, based on analytical proof points.
- By bringing analytics data into the content workflow, editors and marketers can make informed decisions, optimize their online presence in real-time and improve business results.
- It allows marketers to see real-time analytics on the page being worked on.
- Ability to track all relevant information and events related to content, traffic and conversions.
- Predefined analytics best practice guidelines to get the most out of the Episerver platforms.
- Analytics data presented alongside the content being analysed.
- Track the effect of social campaigns on conversions and revenue directly.
- Ability to see the conversions generated from personalization efforts on the site.

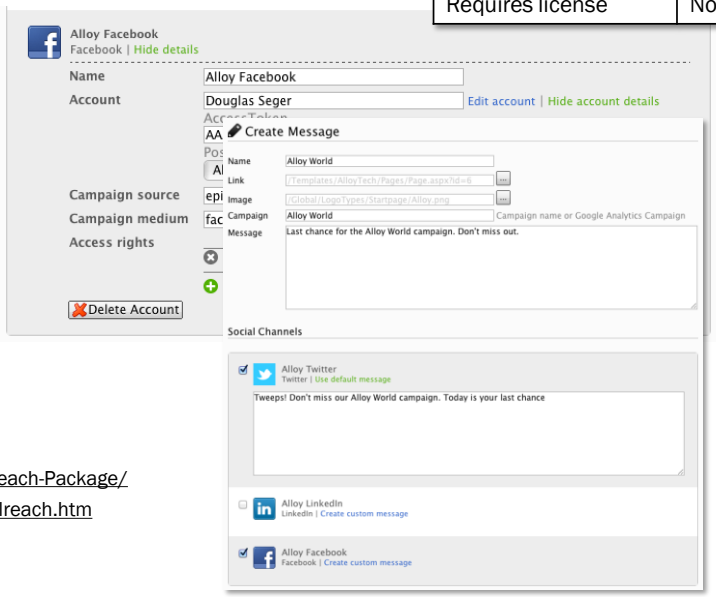
Module F – Extending with Plug-ins and Add-ons – Example add-ons

Supports DXC Service	Yes
Requires license	No

Episerver Social Reach

With **Episerver Social Reach** you can set up your social channels and configure which editors and marketers that are allowed to use them.

When an article or product is to be promoted, create a social message and decide which social channels you want to target it to.



Introducing EPiServer Social Reach

http://world.episerver.com/Articles/Items/Social-Reach-Package/

http://webhelp.episerver.com/latest/addons/socialreach.htm

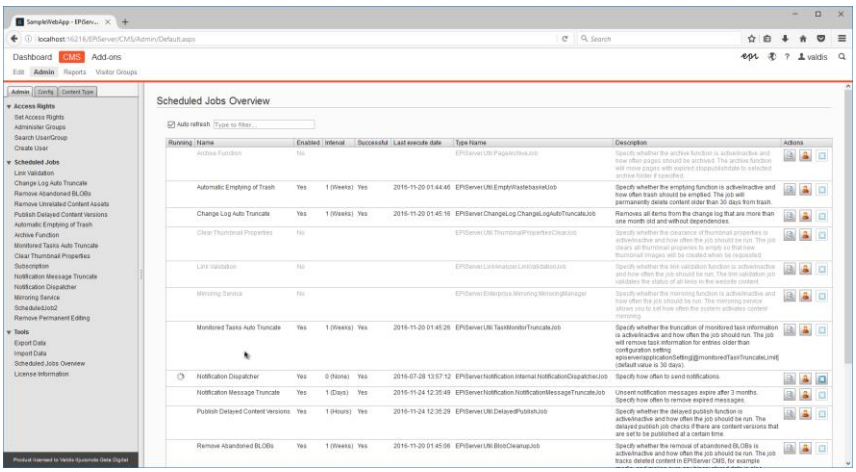
Module F – Extending with Plug-ins and Add-ons – Example add-ons

Supports DXC Service	Yes
Requires license	No

TechFellow ScheduledJobOverview

Gives you an easy way to tell details of the job, which of them is enabled, which of them failed last time, what is the schedule interval, and so on.

Open source on GitHub so you can learn how to build your own Admin view plug-ins.



https://github.com/valdisiljuconoks/TechFellow.ScheduledJobOverview/blob/master/README.md

Module F – Extending with Plug-ins and Add-ons – Example add-ons

Supports DXC Service	Yes
Requires license	Yes

Episerver Social

http://www.episerver.com/services/cloud-service/episerver-social/

User-generated content drives engagement and conversions, and is the most effective way to increase credibility and loyalty with your customers. Episerver Social is the high performance **micro-service** that lets you **store, manage, moderate and deliver ratings, reviews, comments and groups**.

Built on a Data Storage Cluster and Microsoft Azure Service Fabric for robust scalability.

Comments

Moderation

Ratings

Activities

Groups

Do not confuse **Episerver Social** with:
- **Episerver Social Reach**: push messages to Facebook, Twitter, etc.
- **Episerver UGC**: integrate with external social content.

Module F – Extending with Plug-ins and Add-ons – Example add-ons

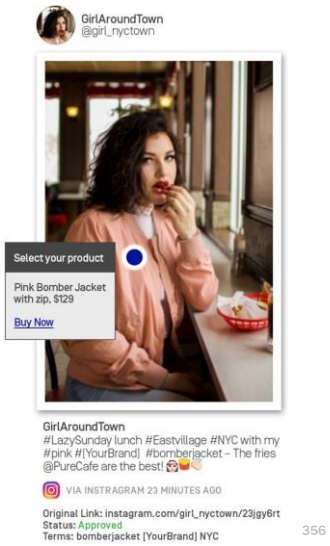
Supports DXC Service	Yes
Requires license	Yes

Episerver UGC

http://www.episerver.com/products/platform/episerver-ugc/

Episerver UGC enables you to display user-generated content on your website, commerce site and other channels. Use your best fan content to turn your website into a highly engaging destination.

- Aggregate, curate and present relevant user-generated content and personalize the experience on your site
- Engage and reward users by spotlighting their content through competitions, interactive maps and visual social galleries
- Drive engagement and grow your fanbase with call-to-action tiles

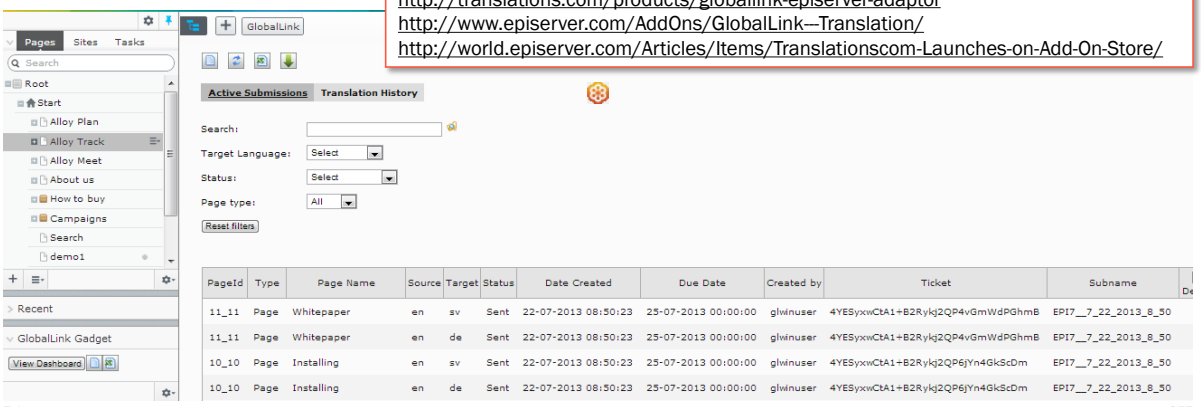


Module F – Extending with Plug-ins and Add-ons – Example add-ons

Supports DXC Service	Yes
Requires license	Yes

GlobalLink® Localization Suite from translation.com

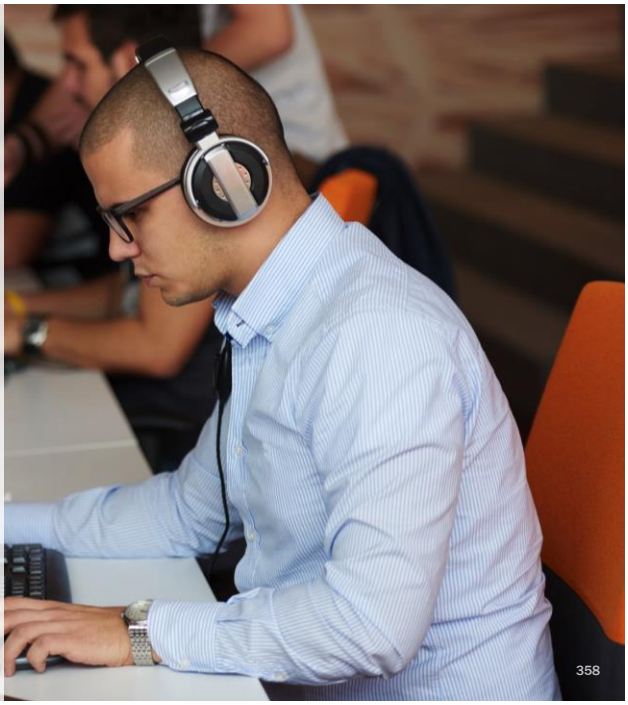
http://translations.com/products/globallink-episerver-adaptor
http://www.episerver.com/AddOns/GlobalLink---Translation/
http://world.episerver.com/Articles/Items/Translationscom-Launches-on-Add-On-Store/



Module F – Extending with Plug-ins and Add-ons

Exercises for Module F

1. Exploring existing add-ons and plug-ins
   Note: one task requires Episerver Find
2. Creating scheduled job plug-ins
3. Creating an admin tool plug-in
4. Creating a report plug-in
5. Integrating with Tasks in the Navigation pane



Episerver
358

Episerver CMS Advanced Development

Module G
Episerver Find

There are several available options when it comes to implementing search solutions for EPiServer sites.
With knowledge of these you can identify the best solution to use based on the requirements for the specific site.

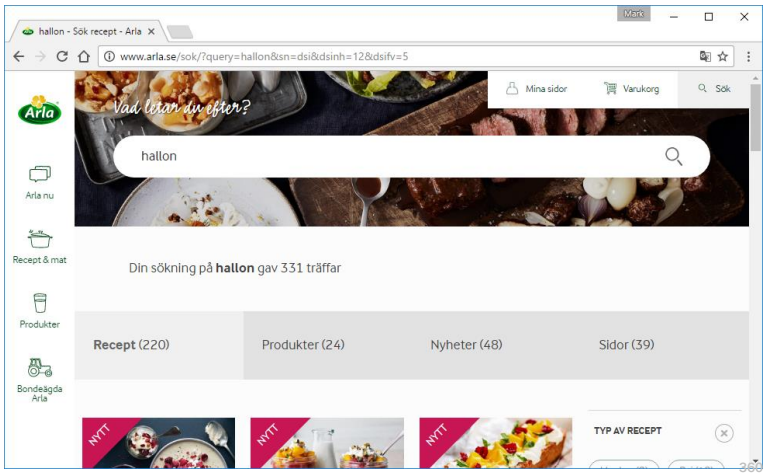
Episerver 359

Sites that use Episerver Find

Arla
http://www.arla.se/

Small Luxury Hotels of the World
http://www.slh.com/

SMALL LUXURY HOTELS OF THE WORLD
Independently minded



Episerver 360

Module G – Episerver Find

Why is Episerver Find better than Elasticsearch?

Episerver Find is based on Elasticsearch, a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements. Learn more: https://www.elastic.co/learn

Why use Episerver Find?

- **Managed Services**: Episerver Find is a SaaS/PaaS cloud solution fully managed by Episerver experts to keep your indexed searches running smoothly.
- **Friendly .NET API**: Episerver Find has an easy-to-understand API that wraps the underlying Elasticsearch REST/JSON service.
- **Integration with Episerver CMS, Commerce, and Personalization**: Episerver Find integrates automatically with our other products, including advanced AI personalization.

Module G – Episerver Find

Episerver Find features

- Multi-language stemming
- Deconstruction of words (Swedish and Norwegian)
- Related queries
- Highlighted summaries
- Autocomplete
- Search as you type
- Search in files/attachments
- Statistics and search optimization
  - Best bets, Custom weighting of results
- Find Connectors

Good Practice
<title> and <meta name="description"> needs to be properly filled for Episerver Find to index an external page correctly. If the meta description is missing, Find will use the nearest <h2> (or <p> tag if <h2> is missing).

Module G – Episerver Find

Language features

Decompounding
- cheeseburger → cheese burger
- football → foot ball
- blårutigskjortan → blå rutig skjorta n (the blue checkered shirt)
- banan → bana n (the trajectory)
- banan → banana

Module G – Episerver Find

Installing Episerver Find

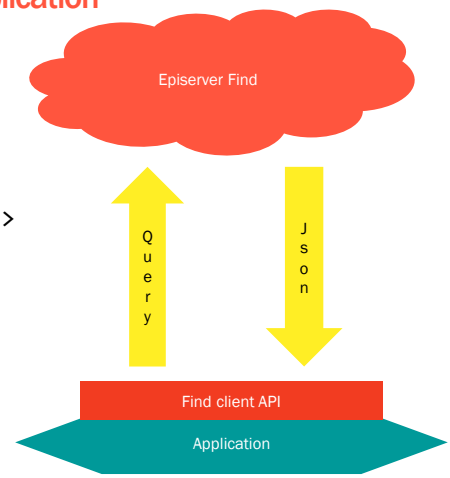
- Installed through NuGet
- Requires additional license + create an index in cloud service
- Support for Episerver CMS 6 and higher
- Support for Episerver Commerce
- Requires the full .NET framework (not Client Profile)
- Depends on JSON.NET (Newtonsoft.Json.dll)

Configuring Episerver Find

```
<configSections>
  <section name="episerver.find"
   type="EPiServer.Find.Configuration,
         EPiServer.Find" />
</configSections>

<episerver.find
  serviceUrl="http://... "
  defaultIndex="myindex"/>
```


Using Episerver Find from any .NET application

```
var result = client.Search<T>
    .For("some search term")
    .GetResult();
```



Module G – Episerver Find

How to create the IClient

```
using EPiServer.Find;
```

Choose one of the following:

- In code with the constructor.

```
IClient client = new Client(serviceUrl: "https://es-eu-api01.episerver.net/Plp...GRv",
    defaultIndex: "episervertraining_index99999", defaultRequestTimeout: 10);
```

- From a configuration file.

```
IClient client = Client.CreateFromConfig(); // any .NET app
```

```
IClient client = SearchClient.Instance; // Episerver site
```

```
<episerver.find
  serviceUrl="https://es-eu-api01.episerver.net/Plp...GRv"
  defaultIndex="episervertraining_index99999" />
```

Episerver 367

Module G – Episerver Find

Indexing documents

```
using EPiServer.Find.Api.Ids;
```

Every document that is indexed is identified by two parts:

- *string* **Type**: the document's type, e.g. **AlloyAdvanced_Models_Pages_StartPage** or **Find4Devs_Book**.
- *DocumentId* **Id**: equivalent to a string of up to 100 characters (no spaces allowed).

The **DocumentId** type has implicit operators that automatically convert from the following .NET types:

- int, Guid, long, DateTime, float, double, and string.

```
DocumentId a = 1;
DocumentId b = Guid.NewGuid();
DocumentId c = DateTime.Now;
DocumentId d = "hello_world";
```

Indexing is done using the client's **Index** method. Any .NET/CLR object can be indexed as long as it can be serialized to JSON.
It's possible to index several objects at the same time using overloads of the Index method which has IEnumerable<object> or params object[] as parameters.

Episerver

What property is used for the Id?

If Episerver Find does not know which property of a type should be used (it does NOT assume one named **Id**), it will generate a GUID you can get from the **IndexResult** return value.

```
using EPiServer.Find.Api;
```

```
namespace Find4Devs
{
  public class Book
  {
    public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string Author { get; set; }
  }
}
```

```
var book = new Book {
  Id = 1,
  Title = "Lord of the Rings",
  Author = "J. R. R. Tolkien"
};
IndexResult result = client.Index(book);
WriteLine($"OK: {result.Ok}, Type: {result.Type}, Id: {result.Id}]");
// => OK: True, Type: Find4Devs_Book, Id: vhMDMCUjQG2E-uxlwvfJuw
```


How to control the property used for the Id

Choose one of the following:

- Apply **[Id]** to the property you want to use.
- Define a convention for instances of the type to specify what the Id is.

```
using EPiServer.Find.ClientConventions;
```

```
client.Conventions
  .ForInstancesOf<Book>()
  .IdIs(b => b.Id);
```

```
namespace Find4Devs
{
  public class Book
  {
    [Id] public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string Author { get; set; }
  }
}
```

```
using EPiServer.Find;
```

```
var book = new Book {
  Id = 1,
  Title = "Lord of the Rings",
  Author = "J. R. R. Tolkien"
};
IndexResult result = client.Index(book);
WriteLine($"OK: {result.Ok}, Type: {result.Type}, Id: {result.Id}]");
// => OK: True, Type: Find4Devs_Book, Id: 1
```

Module G – Episerver Find

Getting by ID

```
var blogPost = new BlogPost
{
      Id = 42
};
client.Index(blogPost);

blogPost = client.Get<BlogPost>(42);
```

Once an object has been indexed it's immediately retrievable by its ID using the clients **Get** method. The Get method requires a type parameter, specifying the type of object to fetch from the index, and the ID of the object.
All objects can be retrieved by ID given that they can be deserialized from JSON.

Module G – Episerver Find

Making changes to documents

Choose one of the following:

- To perform a replacement, i.e. PUT
  - **Get**(): fetch an existing document
  - Modify its properties
  - **Index**(): reindex the document

```
Book book = client.Get<Book>(1);
book.Title = "new title";
IndexResult result = client.Index(book,
  x => x.Refresh = true); // optional
```

- To perform a more efficient update, i.e. PATCH
  - **Update**(): create an updater for an existing document
  - Specify the field to be updated
  - **Execute**() the update

```
ITypeUpdate<Book> updater =
  client.Update<Book>(1);
ITypeUpdated<Book> command =
  updater.Field(b => b.Title, "new title");
IndexResult result = command.Execute();
```

Updating

```
var blogPost = ...
client.Index(blogPost);

blogPost = client.Get<BlogPost>(42);
blogPost.Title = "New title";
client.Index(blogPost);
```

Objects which have been indexed can be updated by indexing them again. The index method does not differentiate between adding new objects or updating existing ones. If an document with the same ID exists in the index it will be overwritten, otherwise a new document will be added.

NOTE!
In the above example we update an object by retrieving it from the index, modifying it and finally indexing it again. Retrieving the object from the index is not a requirement. We could just as well retrieve the latest version of the object from some other data store, such as a database, and index it again to update it in the index. The only requirement is that the ID is the same.

373

Deleting by ID

```
client.Delete<BlogPost>(42);
```

Indexed objects can be removed from the index in several ways. The simplest way is to use the clients Delete method which will delete an object of a specific type with a given ID.

Module G – Episerver Find

Searching

```
var result = client.Search<Book>()
                   .GetResult();
```

Searching is done using the **Search** method. Its type parameter specifies what types to search for. The Search method does not actually perform the search but returns a query object which can be further "configured". If no criteria is added the query will search for all objects of the specified type, including sub types.
The **GetResult** method executes the query, sending it to the server and returning the results. In other words, no communication with the server happens prior to the GetResult method call.

Episerver 375

Module G – Episerver Find

Search results

```
var result = client.Search<Book>()
                   .GetResult();

foreach(var book in results)
{
      Console.WriteLine(book.Title);
}
```

The search results are of a type that implements **IEnumerable** of the searched type, meaning that we can immediately iterate over the returned objects

Episerver 376

Module G – Episerver Find

Search results

```
var result = client.Search<Book>()
                   .GetResult();

int total = result.TotalMatching;

int executionTime =
  result.ProcessingInfo.ServerDuration;
```

By default only the first ten hits are returned. The total number of matching documents can be retrieved using the **TotalMatching** property on the result object.
Information about the execution, such as the number of milliseconds it took to execute the search, on the search engine can be retrieved using the **ProcessingInfo** property.

Module G – Episerver Find

Search results

```
var result = client.Search<Book>()
                   .GetResult();

foreach(var hit in results.Hits)
{
      Console.WriteLine(hit.Document.Title);
      Console.WriteLine(hit.Score);
}
```

The Hits property on the results object contains SearchHit objects. A SearchHit object contains both the matching object as well as metadata, such as the score.

Free text search with For method

```
var result = client.Search<Book>()
    .For("lord of the rings")
    .GetResult();
```

A free text query can be added using the **For** method. In the above example all books with any of the words "lord" and "rings" in any of their properties will be matched.

Warning!
Recently, Episerver changed the default behaviour of the Elasticsearch indexes to remove all stop words, so "of" and "the" are treated the same as "lord" and "rings". If you use the MoreLikeThis() extension method then you can supply a StopWords property with a list of words, but for general queries, remove the stop words using regular expression before running the query. But explicitly Track using the original query text.
https://world.episerver.com/forum/developer-forum/EPiServer-Search/Thread-Container/2017/10/removing-extra-results-that-use-grammatical-article-words/

https://world.episerver.com/forum/developer-forum/Feature-requests/Thread-Container/2017/1/be-able-to-filter-out-stopwords-for-all-search-not-only-morelike/

AND instead of OR

```
var result = client
    .Search<Book>()
    .For("lord of the rings")
    .WithAndAsDefaultOperator()
    .GetResult();
```

When using the **For** method each word in the string passed will by default be "or:ed". Meaning that a string with two words will be interpreted as **<word1> OR <word2>**. Applied to the above example this means that the query would match a book titled "Lord of the flies" as it contains the word "lord".

This is often the desired behavior as a book titled "The lord of the rings" would get a higher score and therefor be placed before Lord of the flies in the results. However, in some cases we may want to limit the search results to such that match all keywords in the query. We can then use the **WithAndAsDefaultOperator** method. For a string with two words passed as argument to the For method will then be interpreted as **<word1> AND <word2>**.

Module G – Episerver Find

Specifying a field to look in

```
var result = client
    .Search<Book>()
    .For("lord of the rings")
    .InField(x => x.Title)
    .GetResult();
```

Using the **InField** method we can specify that the free text query should only be executed against a single field.
Although it's possible to build free text search functionality without specifying what fields to search in it's generally recommended to specify the fields the query should be executed against. By doing so we don't risk exposing anything that we did not intend to.

Module G – Episerver Find

Specifying fields to look in

```
var result = client
    .Search<Book>()
    .For("lord of the rings")
    .InFields(x => x.Title, x => x.Summary)
    .GetResult();
```

Several fields can be specified by either invoking the InField method multiple times or by using the **InFields** method.

Stemming

```
var result = client
    .Search<Book>(Language.English)
    .For("ring")
    .InField(x => x.Title)
    .GetResult();
```

Stemming reduces inflected words to their stem. That is, the words "ring" and "rings" can both be reduced to "ring".
It's important to note that it's not possible to search using stemming in InAllField.
This means that in order for the language parameter to have any effect we must specify what fields to search in using the methods described above, such as InField. Below is an example of a search request for the word "cars" that will match blog posts titled "car" or "A blue car".

Summary: Free text search

- Use the `For` method
- Stemming requires:
  - `Language` argument to the `Search` method
  - One or several fields explicitly specified using `InField/InFields`
- Fields can be boosted but the default behavior usually works well

Module G – Episerver Find

Filtering

```
var result = client
    .Search<Book>()
    .Filter(x =>
        x.Author.Match("J. R. R. Tolkien"))
    .GetResult();
```

When we want to find only documents that matches a specific condition we can use filters. As opposed to free text queries filters either match completely or not at all. That is, while free text queries (and other types of queries) rank documents by score where one document can match the query a lot and another just a little and both are returned, filter does not produce or affect scoring.

The **Filter method** is quite similar to the Where method in LINQ. It does however have a slightly different syntax as it requires an expression that returns a **Filter object** instead of a Boolean value. When using the Filter method we typically use the **Match** method in the filter expression to match a value exactly, or for lists of objects implementing IEnumerable, to match require one of the objects in the list to match a value.

Module G – Episerver Find

String filtering

```
.Filter(x =>
  x.Author.Match("J. R. R. Tolkien")
  x.Author.MatchCaseInsensitive("j. R. r. tOLkien")
  x.Title.Prefix("The lord")
  x.Title.PrefixCaseInsensitive("tHe lORd")
  x.Author.Exists()
  x.Title.AnyWordBeginsWith("rin")
```

NOTE: The AnyWordBeginsWith method while powerful isn't optimal in terms of performance when used for large strings. It's therefore best to limit its usage to short string fields such as titles, names, tags and the like.

String properties can be filtered in a number of ways. For exact matching we can use the Match method and for the equivalent of String.StartsWith we can use the Prefix method. Both methods are case sensitive but have corresponding methods for case insensitive filtering. The Exists method matches properties which have any value.

Numerical filtering

```
.Filter(x =>
  x.NumberOfPages.Match(480)
  x.NumberOfPages.InRange(400, 600)
  x.NumberOfPages.Exists()
```

Numerical values such as integers, doubles, floats and longs as well as their nullable equivalents can be matched by equality using the Match method and for existence using the Exists method. It's also possible to require that a value is within a certain range using the InRange method.

Other built-in filters

- DateTime
- Booleans
- Enum
- Type
- Nested fields
- Collections
- Complex objects
- Negating
- Combined filters

Module G – Episerver Find

The BuildFilter method

```
var filter = client.BuildFilter<Book>();

filter = filter.And(x =>
           x.Title.Prefix("A"));
filter = filter.Or(x =>
           x.Author.Prefix("A"));

client.Search<Book>()
      .Filter(x => filter);
```

Sometimes, especially when reacting to user input, filters has to be dynamically composed. The BuildFilter method can be used to construct a filter which can later be added to a search query.

Module G – Episerver Find

More on filter expression

- Null checks not needed
- Custom filter methods

As filter expressions are not executed "as-is" but parsed and sent over to the search engine we generally don't have to do null-checks like we would with in-memory LINQ queries. For instance, with an expression such as x => x.Author.Prefix("A") it doesn't matter if the Author property has a value or not.
It's possible to extend Finds filtering API by creating custom filter methods. For instance, if we often use x => x.Author.Prefix("A") we could create a method that allows us to instead write x => x.AuthorNameStartsWithA().

Summary: Filtering

- Apply filters to search requests using the `Filter` method
- `Filter` is similar to LINQ's `Where` method but with a different syntax
- Use `BuildFilter` to easily create complex filters based on user input
- Filtering is usually very quick, even on huge data sets

Skip and Take

```
var result = client.Search<Book>()
  .Skip(50)
  .Take(25);
```

Note! Defaults to 10.

Skip bypasses the first n hits that match a search query while Take instructs the search engine to return n number of hits. **Skip** and **Take** are typically used together when presenting search results and listings with paging. **Take** is also often used alone when we're only interested in a limited number of hits.

As opposed to LINQ and most database querying solutions, Find defaults to the equivalent of **Take(10)**. That is, if we don't specify the number of hits to return using Take we'll never get more than 10 hits. Also note that Take will throw an exception if we pass it a value larger than 1000. Should we want all hits we must make multiple requests.
While limiting the result set to the first ten items by default and enforcing a maximum size for result sets may seem odd it makes sense when dealing with search engines. First of all it's very common that we only want a subset of a larger result. Second, large result sets, and "lower" parts of large result sets can be demanding in terms of performance for highly scalable search engines.

Sorting

```
var result = client.Search<Book>()
  .OrderBy(x => x.Title)
  .ThenByDescending(x => x.Author)
```

For sorting, use **OrderBy** and **OrderByDescending**. There are also **ThenBy** and **ThenByDescending** methods which are simply aliases for the two former methods and can be used to make the code more easily readable.

As the sorting is done on the server it's safe to sort on fields that could potentially be null. For instance .OrderBy(x => x.A.B.C) won't cause an exception if either A, B or C are null. Note however that sorting on fields that have never been created can raise exceptions from the search engine. That is, while A may be null in the example, at least one object should have had a non-null value for A. OrderBy orders null values last while OrderByDescending orders them first. This default behavior can be changed by supplying a second argument or type SortMissing. For instance, .OrderBy(x => x.Title, SortMissing.First) will return documents without a Title value first.

Projections

```
var result = client.Search<Book>()
  .For("lord of the rings")
  .Select(x => new
    {
      Name = x.Title,
      Text = x.Summary
    })
```

There are three reasons why we'd use projections:

1. Only the required fields need to be transferred from the search engine server resulting in a smaller response.
2. We can make the result object contain a list of objects tailored for our needs, such as data needed for presentation in a search results listing.
3. Some types may be hard to deserialize from JSON and by using a projection we can work around that. For instance, while Find's Episerver CMS integration supports indexing PageData objects it does not support deserializing them.

Projecting cropped text

```
var result = client.Search<Book>()
  .For("lord of the rings")
  .Select(x => new
    {
      Name = x.Title,
      Excerpt = x.Summary.AsCropped(250)
    })
```

It's possible to use a couple of special methods in projection expressions. One such is the **AsCropped** method which is an extension method for strings. When using this method only the first n characters of the string will be returned from the search engine. The search engine will do it's best to crop at the end of a word.

Facets

- Aggregations based on a search result
- "Piggybacked" with search results
- Many use cases, such as:
  - Number of documents by tag/category
  - Article count per month
  - Number of products per price groups
  - Number of stores 1km from a location, 5 km from..
  - How many documents match a filter

Module G – Episerver Find

Terms facets

```
var result = client.Search<Book>()
  .TermsFacetFor(x => x.Author)
  .GetResult();
```

Perhaps the most common type of facets is terms facets. Terms facets provide a grouping of a specific field within the documents that match a search request. This is typically used to display a list of categories, tags, department names etc. We pass TermsFacetFor an expression to specify what field we want a facet for. The search result will contain a terms facet in addition to the regular search hits. It's possible to customize to request for the facet by passing a second argument to the TermsFacetFor method. By doing so we can specify that the facet should contain more than 10 items: .TermsFacetFor(x => x.Author, x => x.Size = 50)

Module G – Episerver Find

Terms facets

```
var authorCounts = client.Search<Book>()
    .TermsFacetFor(x => x.Author)
    .GetResult();

foreach(var author in authorCounts)
{
  Console.Write(author.Term
    + " (" + author.Count + ")");
}
```

As it's possible to request multiple terms facets within the same search request we must again pass an expression specifying what field the facet is for. The returned object from TermsFacetFor implements IEnumerable<TermCount>. TermCount objects have a Term property, containing the value in the field, and a Count property, containing the number of documents that has that specific value.

Module G – Episerver Find

Unified Search in Episerver Find

Unified Search is a concept in Find's .NET API that allows us to use the common denominator approach without sacrificing type specific querying.

- Easily search over different types by searching for ISearchContent
- Types that don't implement ISearchContent can be included, as if they implemented ISearchContent
- Fields with the same name as a property in ISearchContent can be searched
- Type specific filtering still possible
- Hits are automatically projected to UnifiedSearchHit

Module G – Episerver Find

The Unified Search components

The concept of Unified Search consists of four main parts:

- A common interface declaring properties that we may want to use when building search- (not querying) functionality: ISearchContent.
- An object that maintains a configurable list of types that should be searched when searching for ISearchContent as well as, optional, specific rules for filtering and projecting those types when searching: IUnifiedSearchRegistry, which is exposed by the IClient.Conventions.UnfiedSearchRegistry property.
- Classes for search results that are returned when searching for ISearchContent: UnifiedSearchResults which contains a number of UnifiedSearchHit.
- Special method for building and executing search queries: UnifiedSearch() and UnifiedSearchFor() and an overloaded GetResult() method.

Module G – Episerver Find

ISearchContent:

```
Attachment SearchAttachment { get; }
IEnumerable<string> SearchAuthors { get; }
IEnumerable<string> SearchCategories { get; }
string SearchFileExtension { get; set; }
string SearchFilename { get; set; }
GeoLocation SearchGeoLocation { get; }
string SearchHitTypeName { get; }
string SearchHitUrl { get; }
IDictionary<string, IndexValue> SearchMetaData { get; set; }
DateTime? SearchPublishDate { get; }
string SearchSection { get; }
string SearchSubsection { get; }
string SearchSummary { get; }
string SearchText { get; }
string SearchTitle { get; }
string SearchTypeName { get; }
DateTime? SearchUpdateDate { get; }
```

The ISearchContent interface defines quite a lot of properties allowing it to cover most scenarios when building a search page. Most notable are SearchTitle, SearchText and SearchHitUrl as they are typically the most frequently used when building a search page.

Module G – Episerver Find

UnifiedSearchHit:

```
IEnumerable<string> Authors { get; set; }
string Excerpt { get; set; }
string FileExtension { get; set; }
string Filename { get; set; }
GeoLocation GeoLocation { get; set; }
string HitTypeName { get; set; }
Uri ImageUri { get; set; }
Func<object> OriginalObjectGetter { get; set; }
Type OriginalObjectType { get; set; }
DateTime? PublishDate { get; set; }
string Section { get; set; }
string Subsection { get; set; }
string Title { get; set; }
string TypeName { get; set; }
DateTime? UpdateDate { get; set; }
string Url { get; set; }
```

Similar to ISearchContent, UnifiedSearchHit contains a large number of properties that may be of interest when building a search page. When executing a search using Unified Search fields defined in ISearchContent will be projected to the corresponding properties in UnifiedSearchHit, if they exist.

Module G – Episerver Find

IUnifiedSearchRegistry

Defines the rules for Unified Search

- Lives in **`EPiServer.Find.UnifiedSearch`**
- Accessed by client conventions:
  - **`client.Conventions.UnifiedSearchRegistry`**
- Episerver CMS integration automatically adds ContentData to it
- Only poke around in here if you know what you are doing

IUnifiedSearchRegistry exposes methods for building and configuring a list of types that should be included when searching for ISearchContent. Apart from methods for adding (the Add method) and listing types (the List method) it also declares methods that allow us to add rules for how specific types should be filtered when searching as well as how to project found documents to the common hit type (UnifiedSearchHit).
Unless we want to include some additional type that we cannot add to our own models (for example objects in third party dll:s) or modify the rules for an already added type we typically don't have to care about the registry as the CMS integration will automatically add ContentData to it.

Episerver 403

Module G – Episerver Find

Unified Search – adding (registering) types

```
client.Conventions.UnifiedSearchRegistry
  .Add<Book>();

client.Conventions.UnifiedSearchRegistry
  .Add<Movie>();
```

Types that should be included when using Unified Search must either implement ISearchContent or be registered. To add a type we can use the Add method on the UnifiedSearchRegistry which is exposed as a property on the client's Conventions property. Beyond adding types it's also possible to customize how hits of that type should be projected to UnifiedSearcHit objects. Further, it's also possible to add filters that should be applied specifically for documents corresponding to an added type.

Episerver 404

Unified Search – searching

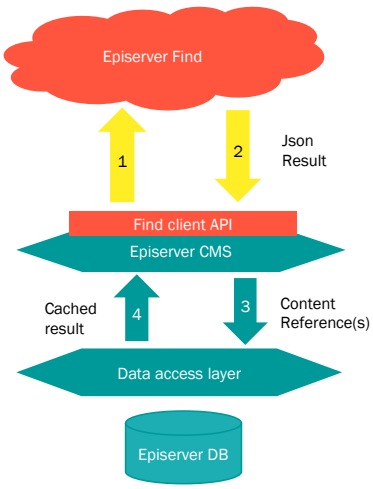
```
client.UnifiedSearch(Language.English)
  .GetResult();
   // is equivalent to
client.Search<ISearchContent>(
  Language.English)
  .GetResult();

client.UnifiedSearchFor(
  “lord of the rings”,
  Language.English)
  .GetResult();
```

To use Unified Search we can either use the standard Search method with ISearchContent as type parameter or use the UnifiedSearch method, both producing the same result.
As Unified Search is primarily geared toward free text search scenarios there is also a method named UnifiedSearchFor which require a string with keywords. When using UnifiedSearchFor a free text search query is added to the search request, similarly to if we had used the For method. UnifiedSearchFor will also automatically specify a number of fields to search in, namely SearchTitle, SearchSummary, SearchText and SearchAttachment.

Using Find with Episerver CMS

```
var client = ServiceLocator.Current
    .GetInstance<SearchClient>();
var result = client
    .Search<IContent>()
    .For("some search term")
    .GetContentResult();
```

When using the Episerver CMS integration to Find, the extension method GetContentResult is available if the type <T> sent to the Search method inherits from IContent.



Obtaining an Instance

- Add a reference to `EPiServer.Find.Framework.dll`
- `SearchClient.Instance`
  - Exposes an `IClient` as a singleton
  - Allows other modules to modify conventions
  - Should always be used if available

Searching for CMS objects such as pages, blocks and assets is done the same way as for any other type of object. That is, we can use the Search method, For method and various other methods described earlier in the course.

Note however that when we're using the integration modules for other Episerver products we should always use SearchClient.Instance instead of creating an IClient using the Client.CreateFromConfig method. The SearchClient.Instance singleton is preconfigured with customized conventions for how CMS types such as PageData and BlockData are serialized and indexed.

Searching

```
var result = SearchClient.Instance.Search<ProductPage>()
                  .For("q")
                  .GetContentResult();

foreach (ProductPage page in result)
{

}
```


Module G – Episerver Find

Client API and CMS - filtering

- Will not automatically filter on access or published

```
SearchClient.Instance.Search<StandardPage>()
    .For("Possibly secret stuff")
    .Filter(x => x.RolesWithReadAccess().Match("Everyone"))
    .GetContentResult();
```

The search engine does not filter documents (such as pages or files) according to access rights. But, if you integrate Find with the Episerver CMS, the return values from the IContent extension methods RolesWithReadAccess and UsersWithReadAccess are indexed. Use these methods to filter content that the current user does not have permission to see.

Module G – Episerver Find

Filtering by part of the content tree

```
SearchClient.Instance
  .Search<IContent>()
  .Filter(x =>
    x.Ancestors().Match(
      PageReference.StartPage.ToString());
```

By default an extension method named Ancestors is included when indexing IContent (pages, shared blocks etc). The Ancestors method returns a list containing the string representation of the ContentLink property of each of the indexed contents ancestors in the content tree. This can be used to filter for content located below a certain node in the content tree.

Filtering by site ID

```
SearchClient.Instance
  .Search<IContent>()
  .Filter(x =>
    x.SiteId.Match("mysite");
```

Episerver Find will automatically index all sites in a multi-site setup but filter the results per-site so that you will by default only get results for the site you are currently browsing.

Unified Search and CMS content

- `PageData` and `MediaData` are added to Unified Search by default
- Default implementations of several `ISearchContent` properties are added for both `PageData` and `MediaData`
  - For example, SearchSection is set to the PageName of the ancestor below the start page and SearchSubSection to the ancestor below the SearchSection page.
- Override or extend by creating properties with matching names in page type classes

Module G – Episerver Find

Indexing content in content areas

While content in a content area is not indexed by default as part of the main content, methods are available for indexing such content.

Use one of the following strategies to index, for example, block type content, inside a content area:

1. Use the content type's **[IndexInContentAreas]** attribute. All instances of the content type that are dropped in a content area are indexed as a part of the main content.
2. In admin mode, create a Boolean property for the content type (selected/not selected) with the name **IndexInContentAreas**, and set its value to True. All instances of that content type in a content area are indexed as part of the main content.
3. Change the default behavior of the **IContentIndexerConventions.ShouldIndexInContentAreaConvention**.

Module G – Episerver Find

Search using Unified Search

```
var result = SearchClient.Instance
    .UnifiedSearchFor("some search term")
    .GetResult();

foreach (UnifiedSearchHit hit in result)
{

}
```

This code will search for IContent and UnifiedFile.
The result object will contain hit objects with a title and an excerpt, both which can be highlighted, as well as some other properties that are commonly used in search result listings.

Module G – Episerver Find

Search statistics and optimization features

Search statistics provide an HTTP API for tracking searches and collecting statistical data about them. It tracks searches by frequency, phrases and number of hits. The aggregated statistics expose functionality you can use to enhance the search: autocomplete, spelling, and related queries.

- Search Statistics support AUTOCOMPLETE suggestions by returning previous queries filtered on a prefix.
- Based on what other users searched for, Search Statistics can provide SPELLCHECKS, popular queries similar to the one passed to the spellchecker.
- Sometimes, it is valuable to discover relationships, for example people who search for a also search for b. Search Statistics calls this RELATED QUERIES.
- Interact with the Search Statistics API using any programming language that makes HTTP requests.

The Framework integration provides an UI to view and work with the statistics features.

Module G – Episerver Find

Expanding Find

- All features are OPT-IN
- Needs to be enabled through code
- For example:
  - Statistics via `Track`()
  - Best Bets via `ApplyBestBets`()
  - Synonyms via `UsingSynonyms`()

Module G – Episerver Find

Personalizing Find

```
private readonly IClient client;
```

How do you make your Find search results personalized? Two method calls:

```
public SearchPageController(IClient client)
{
    this.client = client;
    client.Personalization().Refresh(); // fetch visitor information
```

```
public ViewResult Index(SearchPage currentPage, string q)
{
    var result = client.Search<IContent>()
        .For(q)
        .UsingPersonalization() // personalize query with visitor information
        .FilterForVisitor()
        .GetContentResult();
```


Module G – Episerver Find

Useful links for Episerver Find

```
Install-Package EPiCode.InspectInIndex
```

InspectInIndex

A quick and easy way to inspect Episerver content in an Episerver Find index.

https://github.com/BVNetwork/InspectInIndex/

How to increase the Term Facet Count from default of 10

http://world.episerver.com/forum/developer-forum/EPiServer-Search/Thread-Container/2013/6/Term-facet-count/

Indexing content in a content area

http://world.episerver.com/documentation/developer-guides/find/Integration/episerver-cms-7-5-with-updates/Indexing-content-in-a-content-area/

Searching in blocks

http://world.episerver.com/Modules/Forum/Pages/Thread.aspx?id=65052

Module G – Episerver Find

Learn more about Episerver Find

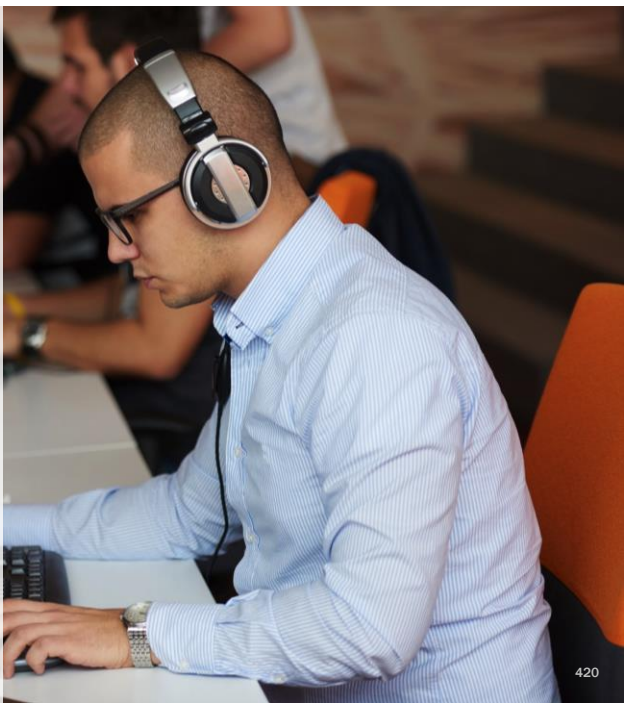
- Training course: Episerver ***Find for Developers*** (1 day)
  http://www.episerver.com/Training/available-courses/developers/episerver-find-for-developers/
- Documentation: http://world.episerver.com/documentation/Other-products/episerver-find
- Forum: http://world.episerver.com/forum/developer-forum/EPiServer-Search/
- Demonstrations:
  - http://flyfind.demo.episerver.com/
  - http://find.demo.episerver.com/

Episerver 419

Module G – Episerver Find

Exercises for Module G

1. Implementing Episerver Find
2. Exploring Episerver Find APIs



Episerver 420

Episerver CMS Advanced Development

Module H
Episerver Social

In this module, you will learn about the Episerver Social cloud service and add-on that developers can use to combine micro-services into advanced, flexible social functions and user-generated content.

Episerver 421

Module H – Episerver Social – Overview

Episerver Social

http://www.episerver.com/services/cloud-service/episerver-social/

User-generated content drives engagement and conversions, and is the most effective way to increase credibility and loyalty with your customers. Episerver Social is the high performance **micro-service** that lets you **store, manage, moderate and deliver ratings, reviews, comments and groups**.

Built on a Data Storage Cluster and Microsoft Azure Service Fabric for robust scalability.

Comments

Moderation

Ratings

Activities

Groups

Do not confuse **Episerver Social** with:
- **Episerver Social Reach**: push messages to Facebook, Twitter, etc.
- **Episerver UGC**: integrate with external social content.

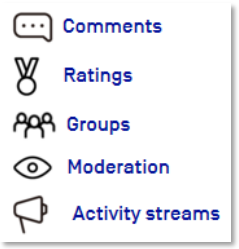
Episerver 422

Module H – Episerver Social – Overview

Episerver Social PaaS for developers

Episerver Social platform is a collection of extensible micro-services for defining and collecting user community generated content.

- **Comments** - manage and deliver hierarchical, user-generated content
- **Ratings** - allow users to quantify the value of your content and products
- **Groups** - aggregate users and content to build digital communities
- **Moderation** - review and control user contributions
- **Activity Streams** - broadcast your audience's engagement with your application



Episerver Social Developer Guide

http://world.episerver.com/documentation/developer-guides/social/

Video (64 minutes): http://fast.wistia.net/embed/iframe/b7x5k8odd4?videoFoam=true

Episerver 423

Module H – Episerver Social – Overview

Episerver Social trial account

Sign in with your Episerver World account to get a free trial of Episerver Social.

Start your Episerver Social trial today!

With your Episerver Social trial, you can begin building social content solutions right now.

- Explore the platform
- Learn to work with the Episerver Social framework
- Build a demonstration application or proof of concept

To start the signup process, please log in with your Episerver World account.

Sign in with Episerver World New to Episerver World?

Episerver 424

Module H – Episerver Social – Overview

Episerver SocialAlloy

SocialAlloy is a clone of the Episerver Alloy sample application, enhanced with components demonstrating Episerver Social:

- To provide a simple application demonstrating Episerver Social features and capabilities
- To provide developers looking to get started with Episerver Social with a helpful point of reference

What's inside?

- Blocks for all social features, for example, CommentsBlock, RatingBlock, LikeButtonBlock, etc.
- CommunityPage: shows examples of: comments, ratings, subscriptions, activities, moderation, etc.
- Moderation user interface

https://github.com/episerver/SocialAlloy

Episerver 425

Module H – Episerver Social – Installing and Configuring

Episerver Social package installation

To install in an Episerver web site, enter the following commands in the Package Manager Console for the features that you want to use:

```
Install-Package EPiServer.Social.Comments.Site -ProjectName AlloyAdvanced
Install-Package EPiServer.Social.Ratings.Site -ProjectName AlloyAdvanced
Install-Package EPiServer.Social.Moderation.Site -ProjectName AlloyAdvanced
Install-Package EPiServer.Social.Groups.Site -ProjectName AlloyAdvanced
Install-Package EPiServer.Social.ActivityStreams.Site -ProjectName AlloyAdvanced
```

Episerver 426

Module H – Episerver Social – Installing and Configuring

Episerver Social configuration

Your application's **appId** and **secret** are private to your application. This information should not be committed to a source control repository or otherwise publicly exposed.

To configure Episerver Social, copy and paste from the email you were sent for your account:.

```
<episerver.social>
  <settings timeout="100000"/>
  <authentication appId="your-application-id" secret="your-application-secret"/>
  <endpoints>
    <add name="Comments" value="https://..." />
    <add name="Ratings" value="https://..." />
    <add name="Moderation" value="https://..." />
    <add name="ActivityStreams" value="https://..."
    <add name="Groups" value="https://..." />
  </endpoints>
</episerver.social>
```

It is essential that the server hosting your application maintains accurate time. When the server time is inaccurate, requests are created with inaccurate timestamps. As a result, these requests may be rejected as unauthentic.

Module H – Episerver Social – Common patterns

Episerver Social services

Each service implements an interface:

- **ICommentService**, **IRatingService**, and so on

To get an instance inside an Episerver web site, use dependency injection, for example:

```
ICommentService commentService =
    ServiceLocator.Current.GetInstance<ICommentService>();
```

Module H – Episerver Social – Common patterns

Episerver Social exceptions

Common exceptions thrown include:

- **SocialAuthenticationException**: misconfiguration, server time out-of-sync, and so on.
- **MaximumDataSizeExceededException**: if social content is more than 10 kilobytes in size.
- **RateLimitExceededException**: if you issue too many requests over a short period of time.
- **SocialCommunicationException**: if an application cannot connect or communicate with Episerver Social platform cloud services.
- **SocialException**: unexpected errors.

The individual services may also throw exceptions that are unique to the feature that they support.

Module H – Episerver Social – Common patterns

Episerver Social IDs and references

IDs are for the entities of an Episerver Social feature.

- The values are internally-generated and used to distinguish individual entities within the system.
- The classes are **CommentId**, **GroupId**, and so on.

References are for users or resources *outside* the Episerver Social platform, including content in Episerver CMS and Commerce.

- The value is defined by the developer.
- A URI or similar namespace scheme provides an ideal template for a reference. The following is an example of a reference scheme that might be applied to Episerver Commerce content:

```
resource://episerver/commerce/{product-identifier}/{variant-identifier}
```

Module H – Episerver Social – Common patterns

Episerver Social composites

All Episerver Social features distil social concepts to their essence and allow its native entities to be composed with custom data models for extensibility.

Extension data is a .NET class, defined within your application, intended to capture additional details necessary to shape a platform entity to meet your application's needs.

The platform's services encapsulate the relationship between their entities and extension data with the **Composite** class. The Composite class represents a simple pairing, an instance of a native platform entity and its associated extension data.

Extending comments with composites

http://world.episerver.com/documentation/developer-guides/social/comments/extending-comments-with-composites/

Module H – Episerver Social – Common patterns

Episerver Social criteria for retrieving result sets

These services accept criteria that dictate how to retrieve a result set.

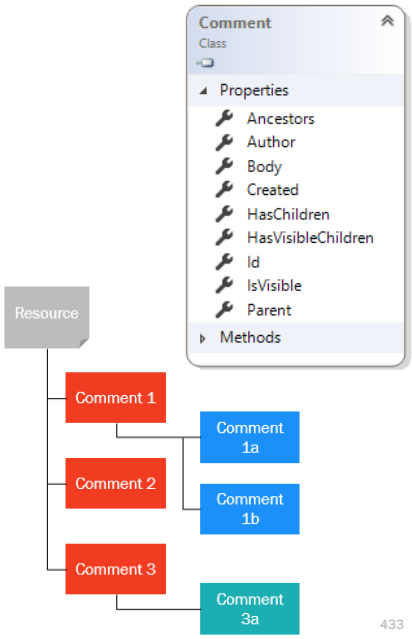
- A class named **Criteria<TFilter>** encapsulates the specifications necessary to retrieve a collection of results from one of the platform services.

Criteria

http://world.episerver.com/documentation/developer-guides/social/social_platform-overview/discovering-the-platform/#criteria

Module H – Episerver Social – Comments

Episerver Social – Comments



Comments are hierarchical in nature, so share relationships with resources and other comments.

A comment has a parental relationship. The parent of a comment is the entity to which the comment replies. That entity may be a resource, such as content or a product, or another comment.

Managing comments

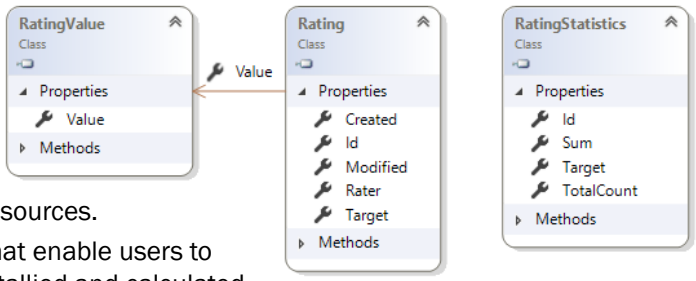
http://world.episerver.com/documentation/developer-guides/social/comments/managing-comments/

User-generated content for ecommerce: reviews and beyond

http://www.episerver.com/learn/resources/blog/adam-blomberg/user-generated-content-for-ecommerce-reviews-and-beyond/

Module H – Episerver Social – Ratings

Episerver Social – Ratings



Ratings let users quantify the value of content, products, and other application resources.

You, as a developer, can design features that enable users to provide quantifiable feedback that can be tallied and calculated, producing meaningful measures to appraise that content.

The value of a rating is represented as a simple integer value. The value's significance is defined in your application.

- A simple 5-star scale might be represented by values 1-5
- A 5-star scale, allowing half-star ratings, might be represented by values 1-10
- A percentage-based scale might be represented by values 1-100

Managing ratings http://world.episerver.com/documentation/developer-guides/social/ratings-intro/managing-ratings/

Module H – Episerver Social – Groups

Episerver Social – Groups: roles, associations, and membership

Groups allow you to combine users and content to create digital communities.

- **Roles** provide a means of labelling or categorizing members within your digital community. They are defined, within your application, as you see fit. They may be assigned to members of a group or span multiple groups. Roles do not bestow any particular permission, status, or responsibility. This leaves your application free to apply meaning to roles as appropriate.
- You **associate** resources with a group by adding them as an association.
- Users are associated with a group by adding them as a member.

Managing groups, roles, associations, and membership

http://world.episerver.com/documentation/developer-guides/social/groups/managing-roles/
http://world.episerver.com/documentation/developer-guides/social/groups/groups-content-associations/
http://world.episerver.com/documentation/developer-guides/social/groups/groups-membership/

Module H – Episerver Social – Groups

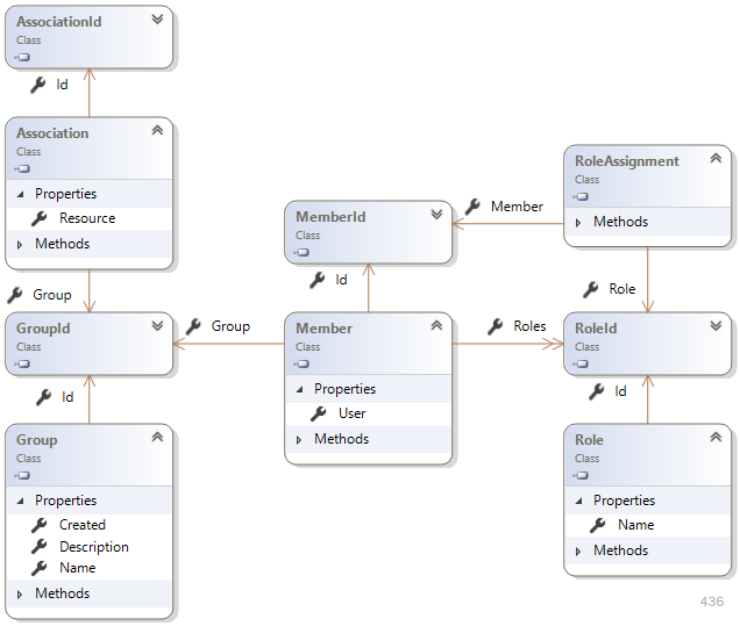
Episerver Social – Groups: roles, associations, and membership

Groups can:

- Have many members
- Have many **associations** with resources e.g. content

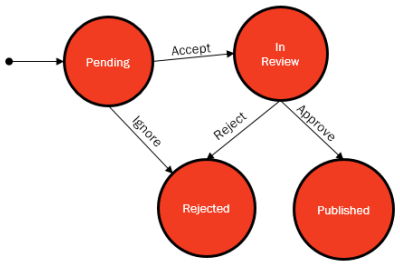
Members can :

- Reference one user e.g. Alice
- Belong to one group
- Belong to many **roles** e.g. Forum Moderator



Module H – Episerver Social – Moderation

Episerver Social – Moderation



Moderation is a business process by which resources and actions may be reviewed for suitability within an application.

- Resources may exist within or outside of the social platform. So, the feature lets you moderate custom resources, such as comments, ratings, profile images, and products.
- Actions represent an activity or request within your application, such as a request to join an exclusive group or publish a comment.

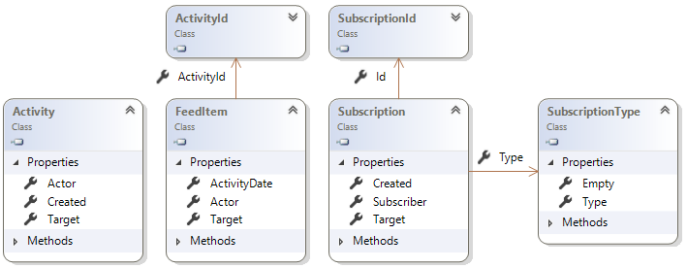
As you plan a moderation strategy, it is important to consider:

- What you intend to moderate (a resource, an action, or a custom entity)
- The steps or process required to moderate it
- How to represent entities being moderated

http://world.episerver.com/documentation/developer-guides/social/moderation-intro/

Module H – Episerver Social – Activity Streams

Episerver Social – Activity Streams



Activity Streams allows developers to:

- Manage subscriptions to resources and other users
- Define and broadcast activities
- Filter and retrieve a feed of information about activities occurring in the application
- React to those activities

A user may subscribe to resources or other users within your application. When that occurs, the system generates a record of activities related to those resources and users. That information can subsequently be filtered and retrieved in the form of a feed.

Activity Streams: subscriptions, feeds, activities

http://world.episerver.com/documentation/developer-guides/social/activity-streams-introduction/

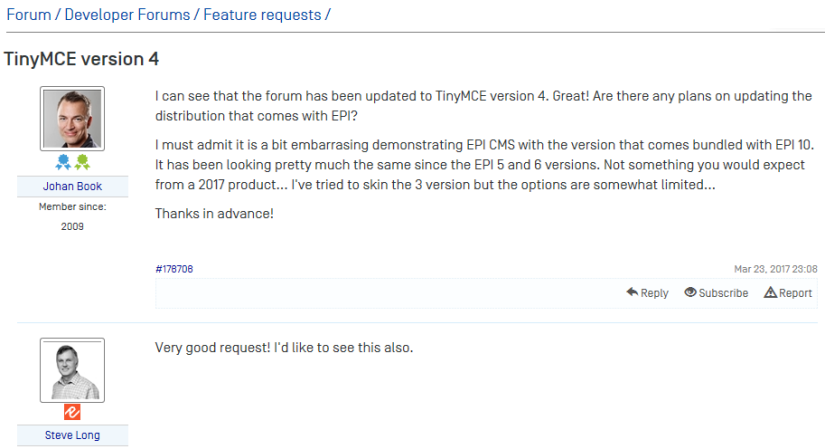
Episerver Social – combining micro-services for Forums

Episerver World has forums, for example, **Developer Forums**, including one for **Feature requests**.

Note that members can:

- **Reply** to a post.
- **Subscribe** to a post.
- **Report** a post.

Members of the forum have a picture, name, and badges.



Module H – Episerver Social – Combining microservices

Episerver Social – combining micro-services for Forums

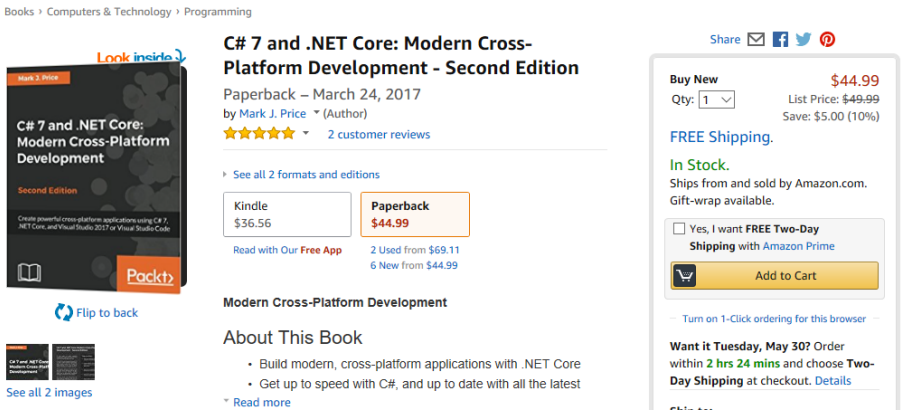
To implement forum functionality on your web site similar to Episerver's, you could combine all Episerver Social's micro-services:

- **Comments**: hierarchy of posts and replies.
- **Ratings**: combine with post or reply to create a "report". If more than one member reports a post, perhaps it is temporarily hidden and flagged for forum administrator review.
- **Groups**: use groups for Members and Moderators. Members could have extended data like badges for Episerver Certified Developers, and Episerver employees using Episerver Social composites.
- **Moderation**: use to determine membership of forums, and special badges to show.
- **Activity Streams**: allow members to see posting activity so they can get answers to their questions ASAP.

Module H – Episerver Social – Combining microservices

Episerver Social – combining micro-services for Reviews

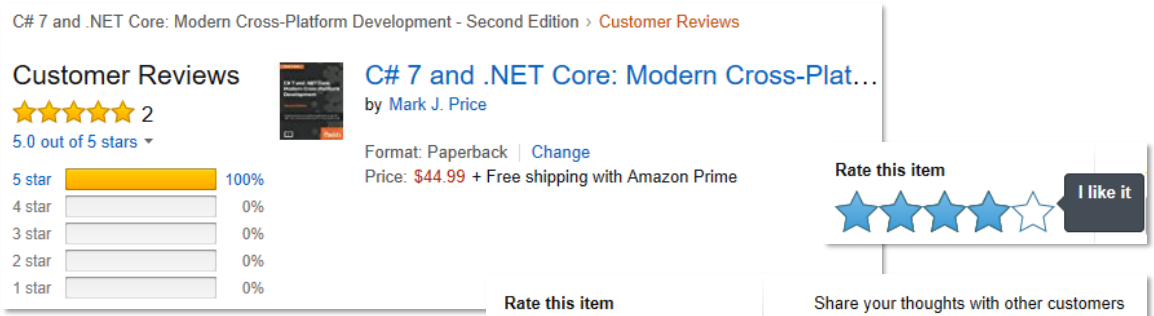
A common feature of web sites is the ability to provider reviews, like on Amazon.



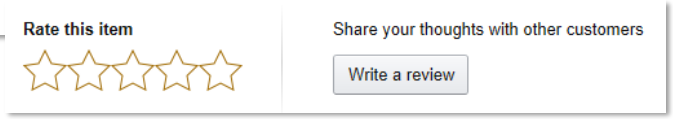
Module H – Episerver Social – Combining microservices

Episerver Social – combining micro-services for Reviews – summary, rate, and review

Review summary shows average of ratings and a break down by rating:



User can add a rating and write a review:



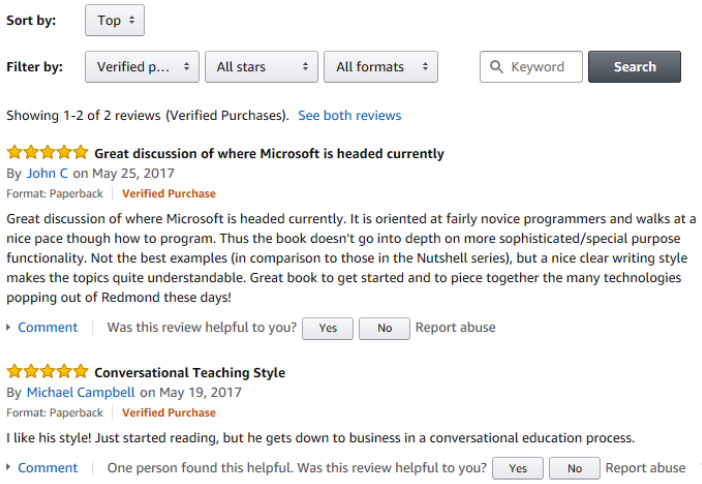
Episerver Social – combining micro-services for Reviews – list of reviews

The review list is:

- Sortable, Filterable, Searchable

Each review has:

- Rating and Title
- Author and Date
- Verified purchase
- Comment (and ability to respond)
- Ability to rate the review as helpful and report abuse

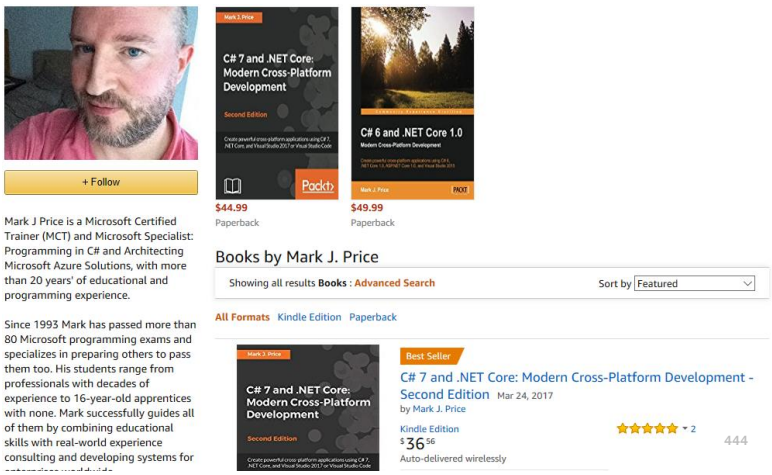


Module H – Episerver Social – Combining microservices

Episerver Social – combining micro-services for Reviews – author page

Authors can register with Amazon and once they are confirmed as the author of a book (or two), they can manage their own page.

- Customers can follow the author to be notified of new publications.
- Authors can write a biography, and manage their list of books.



Episerver Social – combining micro-services for Reviews

To implement functionality on your web site similar to Amazon's, you could combine all Episerver Social's micro-services:

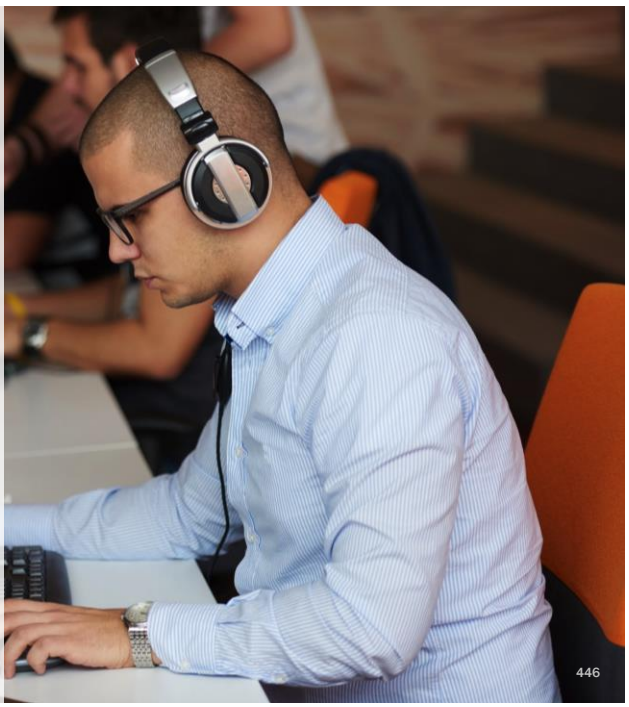
- **Comments**: hierarchy of reviews and responses.
- **Ratings**: combine with comment to create a "review", or individual rating without review text; use built-in aggregation feature to show summary.
- **Groups**: use groups for VerifiedPurchaser and Author. Authors can have extended data like Biography using Episerver Social composites.
- **Moderation**: use workflow to determine membership of VerifiedPurchaser and Author groups.
- **Activity Streams**: allow authors to see reviewing activity so they can respond ASAP, and allow customers to follow the author.

Exercises for Module H

The following exercises require an Episerver Social account:

1. Exploring the SocialAlloy demo site

Apply for a free Episerver Social trial account:
http://demo.social.episerver.net/



Episerver CMS Advanced Development

Conclusion

Episerver 447