Episerver CMS

Advanced Development


Episerver CMS
Advanced
Development
http://www.episerver.com/training/available-courses/developers/episerver-cms-advanced-development/
08 February 2017
Episerver

Episerver CMS – Advanced Development
Course code: 170-3030
Product version: 10.3
Course material version: 10 revision A
Course material date: 2017-02-08

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 CMS Development Fundamentals or equivalent experience / self-studies.

Introduction

Topics

- Why use a CMS?
- Product history
- About the course
- Getting more information
- Continuous delivery
- System requirements
- Tips

Introduction

Why use a CMS?

What are the benefits of using a Content Management System?

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 retain multiple versions of content.

Introduction – Product history

Episerver CMS – the first decade

- Development started 1996
- **Version 1**, release 1997
- **Version 2**, release 1998
  - Inline rich text editor
- **Version 3**, release 2000
  - Frameless
  - Customizable
- **Version 4**, .NET platform, release 2002
  - Version 4.60, support for both ASP.NET 1.1 and 2.0

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 CMS 6 R1 – 2010

- 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)
  - PropertySettings
  - OnlineCenter
    - Gadgets
  - Dynamic Data Store (DDS)
  - Mirroring 2.0

Introduction – Product history

Episerver CMS 6 R2 – 2011

- Released in April 2011
  - Visitor Groups and Personalization
  - On Page Edit is cross-browser compatible
  - Improvements to the Dynamic Content feature
  - 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/

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/

Introduction – Product history

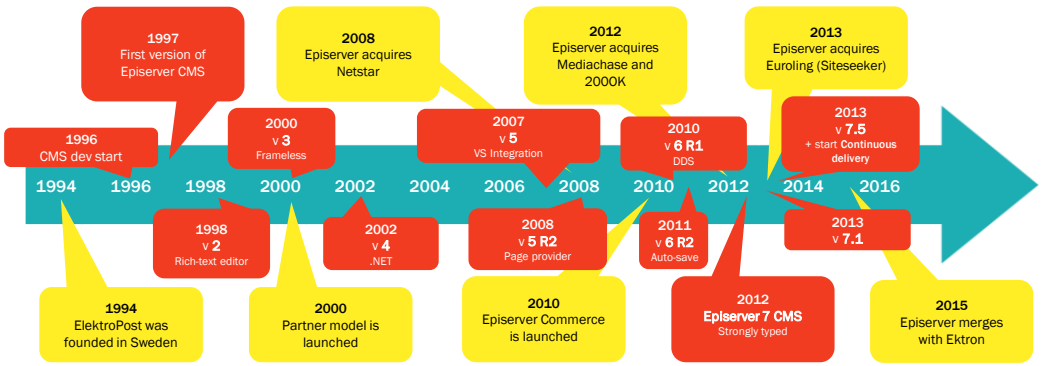
Episerver CMS – 2015 to present

- **Version 8.0, 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 9.0, 22 September 2015**
  - http://world.episerver.com/releases/episerver--update-81/
- **Version 10.0, 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/

Episerver

Introduction – Product history

Key milestones for Episerver CMS



Episerver

Episerver CMS 10

This course covers Episerver CMS version 10. Ted Gustaf has a good blog post about how to upgrade:
https://tedgustaf.com/blog/2016/upgrade-to-episerver-10/

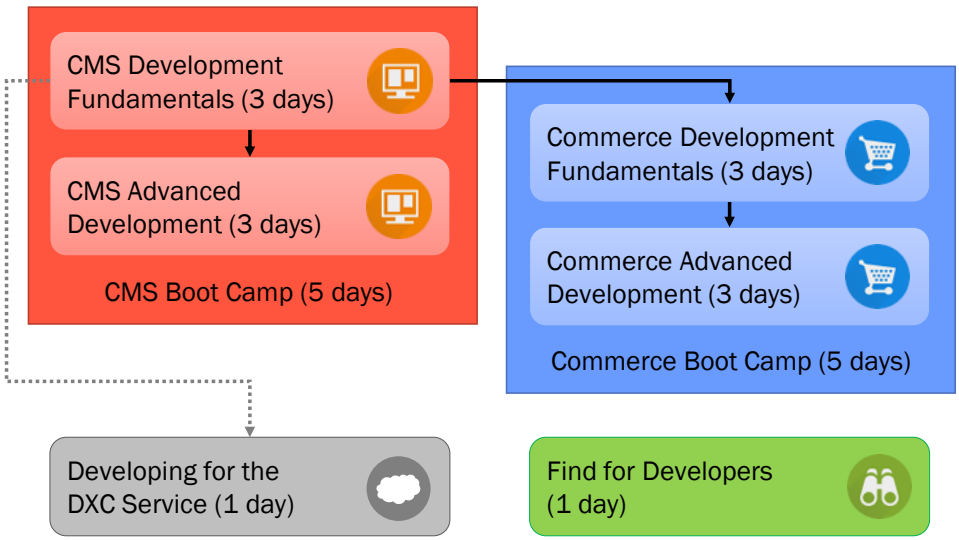
24th October 2016: CMS 10, Commerce 10, and Forms 4 were released.

Breaking changes in CMS 10:
http://world.episerver.com/documentation/upgrading/Episerver-CMS/10/breaking-changes-cms-10/
Breaking changes in Commerce 10:
http://world.episerver.com/documentation/upgrading/episerver-commerce/commerce-10/breaking-changes-commerce-10/
Breaking changes Episerver Forms 4.0:
http://world.episerver.com/add-ons/episerver-forms/breaking-changes-episerver-forms-4/

Course tracks

Education Services offers developer course tracks.

To attend this course you should have skills equivalent to *CMS Development Fundamentals*.



Introduction – About the course

Course Agenda

Day 1
- **Introduction**
- **Module A: Episerver CMS System Review**
- **Module B: Working with Content using APIs**

Day 2
- **Module C: Customizing Properties**
- **Module D: Integrating Data**
- **Module E: Rendering and Personalizing Content**

Day 3
- **Module F: Localizing for Editors and Visitors**
- **Module G: Indexing Content with Search and Find**
- **Module H: Extending with Plug-ins and Add-ons**

Introduction – About the course

About the course exercises

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.

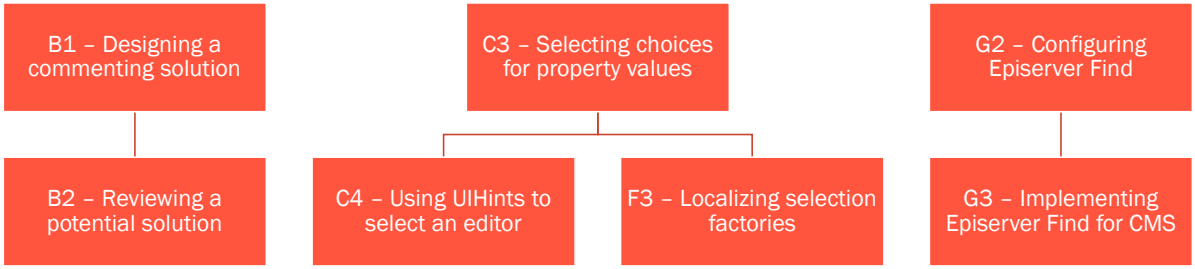
We picked the Alloy reference site as a starting point because
- It is built-in with the Episerver CMS Visual Studio extension,
- It is quick to set up with some sample content,
- It is small enough to understand, and familiar to many Episerver developers, and
- It shows some good practices.

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

Exercise dependencies

All the exercises are designed to work stand-alone as much as possible. All exercises are dependent on the completion of **Exercise 0**, that sets up an **Alloy (MVC)** site with updated packages and database.

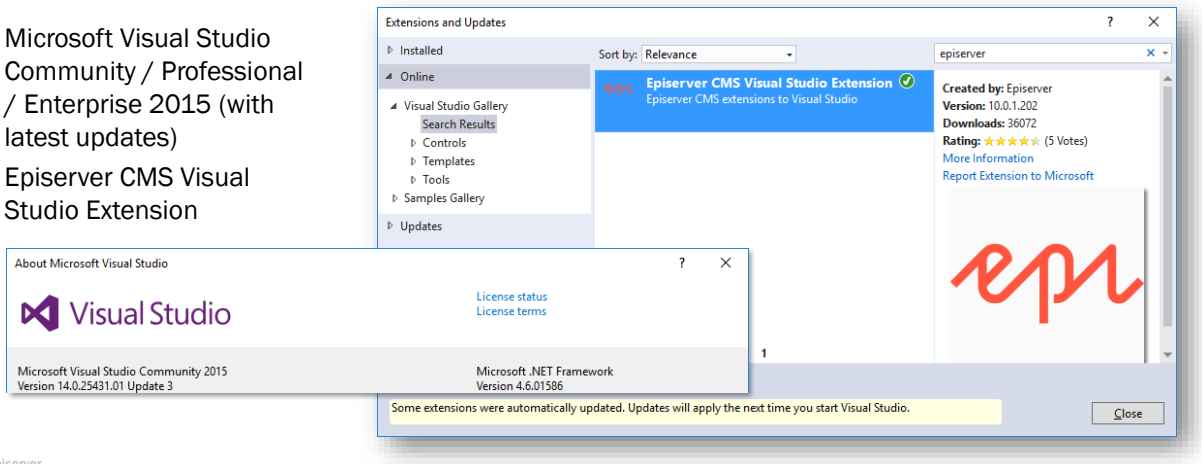
The only other dependent exercises are:



Introduction – About the course

Software requirements

- Microsoft Visual Studio Community / Professional / Enterprise 2015 (with latest updates)
- Episerver CMS Visual Studio Extension



Demo sites

We have a set of public demo sites available demonstrating possibilities with the Episerver platform.
Feel free to explore and use any of these sample websites.
Be aware that the demo sites are refreshed and reset on a daily basis.
You'll find links to presentations about the demo sites with scenarios and logins.
http://world.episerver.com/download/demo-sites/

The following link has a version of the Alloy reference site containing additional features for demoing purposes.
https://github.com/episerver/AlloyDemoKit

Introduction – Getting more information

Knowing where to get help

Where would you go to:

- Read the official documentation for CMS developers?
  - http://world.episerver.com/cms
- Ask a question not in the documentation?
  - http://world.episerver.com/forum
- Record a support ticket?
  - http://world.episerver.com/support
- Find a list of fixed bugs and information about a specific release?
  - http://world.episerver.com/releases
  - http://world.episerver.com/documentation/Release-Notes/

Introduction – Getting more information

Where to go for information

Episerver World: http://world.episerver.com/

- Technical notes, Release notes, Downloads, Developer Forums, Blogs
- Developer Guide with documentation, how-tos, and samples.

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

- Typography, Colors, Icons, Buttons

Episerver on GitHub: http://www.github.com/episerver/

Introduction – Continuous delivery

What does continuous delivery mean?

From version 7.5, Episerver has implemented continuous delivery. This means:

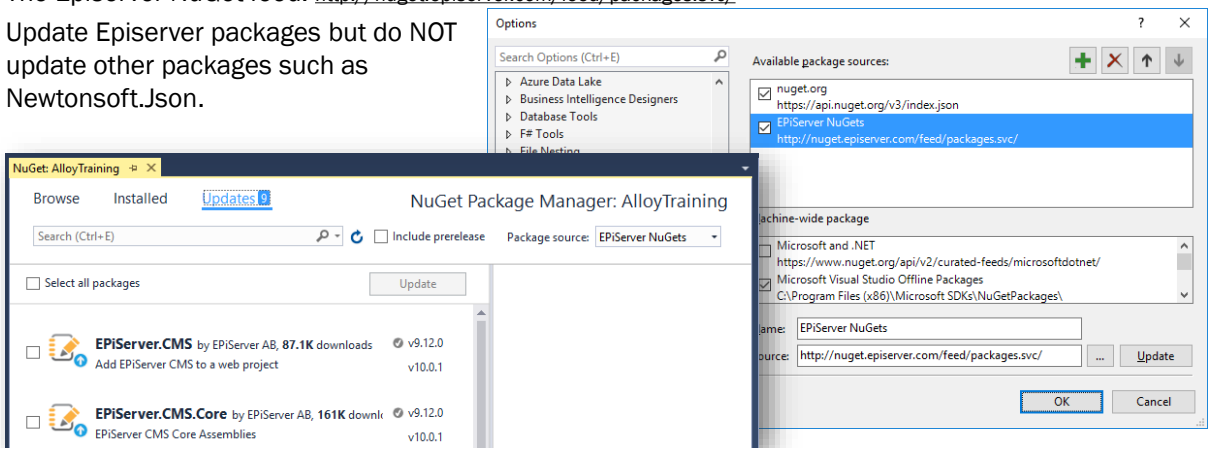
- Weekly updates for all Episerver products.
- Market releases a few times every year that summarize the updates since the previous market release, with mainly a user interface focus.
- Installation via NuGet only; Deployment Center is gone.
- Installed version can be seen in Admin on the browser tab and in the Plug-in Manager.
- One licence file, licence.config, that covers all products.
- Customers are recommended to keep their environment up-to-date.

Episerver NuGet Feed Explorer
Filter by creator e.g. Episerver or other
http://www.david-tec.com/episerver-nuget-feed-explorer/

Updating the NuGet packages

The Episerver NuGet feed: http://nuget.episerver.com/feed/packages.svc/

Update Episerver packages but do NOT update other packages such as Newtonsoft.Json.



Note that you can also use a Web.config setting:

```
<episerver.framework
  updateDatabaseSchema="true"
  createDatabaseSchema="true"
```

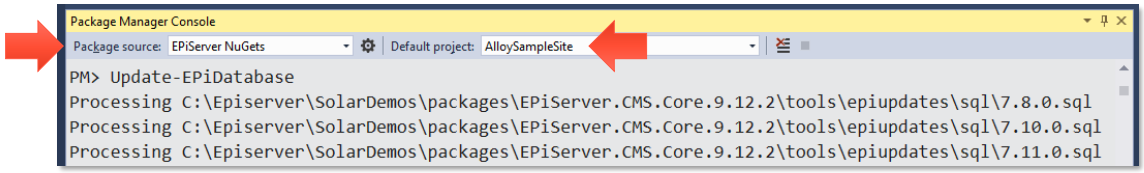
Updating the Episerver database

After updating packages, if you run the site, you might see an exception like this:

Server Error in '/' Application.

The database 'EPiServerDB' has not been updated to the version '7036.0', current database version is '7032.0'. Update the database manually by running the cmdlet 'update-epidatabase' in the package manager console or set 'updateDatabaseSchema="true"' on episerver.framework configuration element

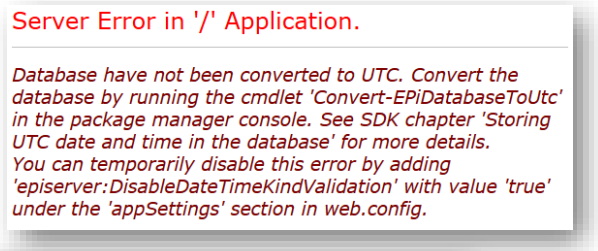
In the Package Manager Console, select the correct **Package source** and **Default project**, and then enter the following PowerShell Cmdlet: **Update-EPiDatabase**

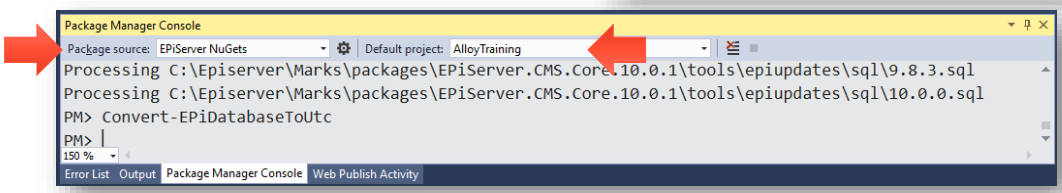


Introduction – Continuous delivery

Updating the Episerver database to use UTC dates and times

After updating the Episerver database, if you run a CMS 10 site, you will see this exception if you were updating from a pre-10 site. In the Package Manager Console, select the correct **Package source** and **Default project**, and then enter the following PowerShell Cmdlet: **Convert-EPiDatabaseToUtc**





Episerver

Introduction – System requirements

Production system requirements

- What are the operating system and .NET platform requirements for a *production* server?
  - Windows Server 2012 or 2012 R2 with IIS 7.0 or later, and
  - Microsoft .NET Framework 4.5.2 or any later compatible version.
- Can you use Oracle as the database?
  - No. Only SQL Server 2008 or later is supported.
- Which browsers are supported for *editors and admins* (NOT visitors)?
  - IE 10 and 11, or latest versions of Chrome and Firefox.

Read the detailed system requirements:
http://world.episerver.com/documentation/Items/System-Requirements/system-requirements--episerver/

Episerver

DXC Service system requirements

- Can you use the same products, features, and add ons, when hosting Episerver CMS in the DXC Service (cloud) as when hosting on your own Windows server(s)?
  - No. CMS Search, mirroring, and some add-ons such as CMO and Social Reach or not supported.

Read the DXC Service system requirements:
http://world.episerver.com/digital-experience-cloud-service/requirements/

Troubleshooting tips for Visual Studio, part 1

Try closing and re-opening views (.cshtml)
Sometimes Visual Studio mistakenly shows errors in views even after the error has been fixed.

Disable Web Sockets for real-time communication on Windows 7

```
<add key="Epi.WebSockets.Enabled" value="false" />
```

Disable Visual Studio's Browser Link
If the browser seems to hang while drawing the Episerver UI, it may be Visual Studio's Browser Link feature interfering with our Dojo library: disable Browser Link to prevent the JavaScript error.

Disable ASP.NET's optimized compilations
If you get ASP.NET dynamic compilation errors, disable **optimizeCompilations** in the root Web.config:

```
<system.web>
    <compilation debug="true" targetFramework="4.5.2" optimizeCompilations="false" />
```

Once it's working again, reset back to `false` for faster performance. ☺

Introduction – Tips

Troubleshooting tips for Visual Studio, part 2

Try closing and re-starting Visual Studio
Sometimes Visual Studio gets confused. Exiting and re-starting sometimes fixes it.

Reset IIS Express

Use the `iisreset` command line to stop and restart IIS Express.

Empty ASP.NET Temporary Files folder

By default, the dynamic compilation of views stores the assemblies (*.dll) in a temporary folder. Stop the site, shut down Visual Studio, and clear the folder sometimes fixes issues.

Episerver

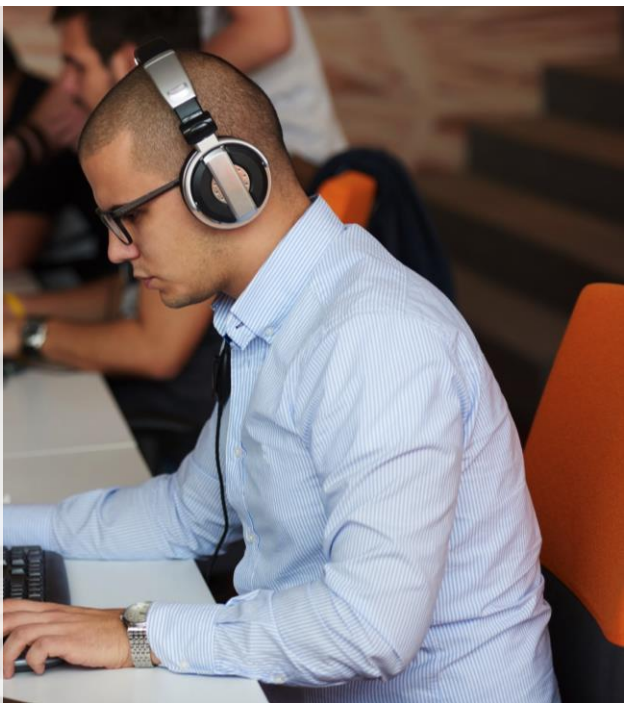
Introduction

Exercise 0

Setting up the AlloyTraining site

In this exercise, you will set up an Alloy sample site ready to extend during the rest of the exercises.

Prerequisites: Microsoft Visual Studio 2015 with Episerver CMS Visual Studio Extension.



Episerver

Episerver CMS System Review

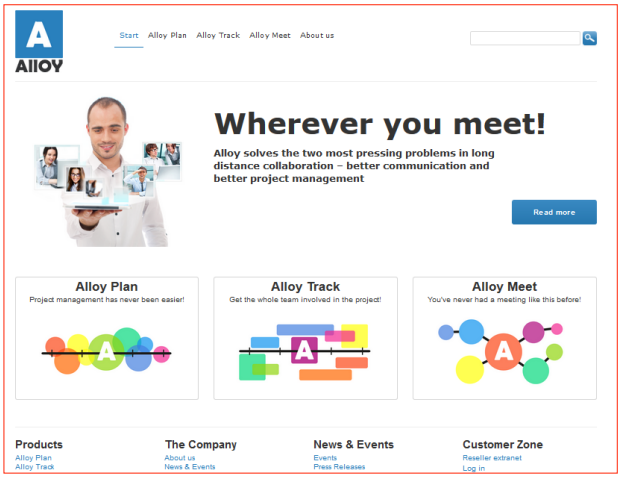
Topics

- Planning the site
- Customizing for the editor
- Architecture and API Overview
- Initialization
- Logging
- Security
- Caching
- Good practice

Episerver

Episerver CMS System Review – Plan the site

Identify parts



Logo

Top menu

Search field

Large teaser

Small teasers

Page footer

Episerver

Episerver CMS System Review – Plan the site

Blocks

- What is the difference between a block that used as a property and a block that is shared?
- TeaserBlock: Same teaser is used on several pages.
  - Shared block, added to ContentArea
- LogoBlock: One block used on every page
  - Block used as property
- What is the difference between For All Sites, For This Site, and For This Page / Block?
- For This Site is not visible by default. How is it activated?
- Where should site settings be stored?
- Page list: Used on every page, just one property
  - No block, settings properties

Episerver

Episerver CMS System Review – Plan the site

Block types

- Both TeaserBlock and LogoBlock have Image properties. Why does one use ContentReference and the other use Url? What's the difference?
- **TeaserBlock**
  - Heading: string
  - Text: string
  - Image: ContentReference, reference to site content
- **LogoBlock**
  - Image: Url
    - With fallback to default image
  - Title text: string

Episerver

Episerver CMS System Review – Plan the site

Page types and common parts

- Look around on the site and find types of pages
  - Page types
- Find parts common between page types
  - Partials (also called "page partials")
  - Media types

Episerver

Episerver CMS System Review – Plan the site

Common Episerver design patterns and conventions (1 of 2)

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 types named PageData and BlockData. Create derived types named SitePageData and SiteBlockData with properties that will be common to all pages and blocks on your site.

Episerver has an attribute named ContentTypeAttribute. Create a derived type named SiteContentTypeAttribute that has a default constructor that sets the GroupName to “Default” (or use a constant). Apply this attribute to your page type classes.

Episerver has an attribute named ImageUrlAttribute. Create a derived type named SiteImageUrlAttribute that has a default constructor that sets the path to the default template image file. Apply this attribute to your page type classes.

Episerver CMS System Review – Plan the site

Common Episerver design patterns and conventions (2 of 2)

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

Note: Alloy uses PageControllerBase<T> which isn’t consistent with the convention but it is commonly used in .NET Framework.

Episerver CMS System Review – Plan the site

Data that is not content

- Can we find anything in Alloy other than editorial content?
  - Editorial content includes: pages, blocks, media assets.
- Where would you store data?
- Use Dynamic Data Store to store simple custom data, for example, XForm data and Visitor Groups statistics.
  - No user interface for editing, although a custom user interface to DDS data could be built.
  - Data that needs to be edited is usually not suited for the DDS.
- Use RDBMS or NoSQL store for more complex or relational data.

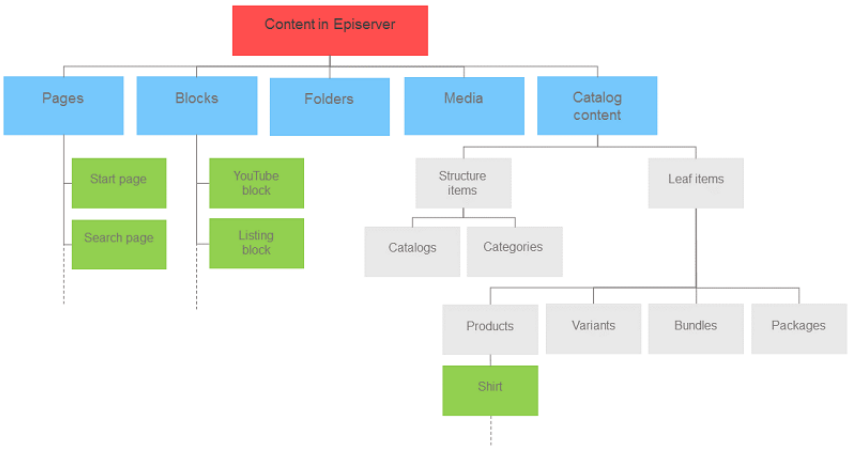
Episerver CMS System Review – Customizing for the editor

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.
  - Plug-ins extend the user interface, e.g. Versions gadget.
  - Custom properties/property types.
  - UIHint: modify the way a property is displayed to the editor, e.g. UIHint.Image.
  - Custom validation: implement IValidate<T>
  - 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.

Episerver CMS System Review – Architecture and API overview

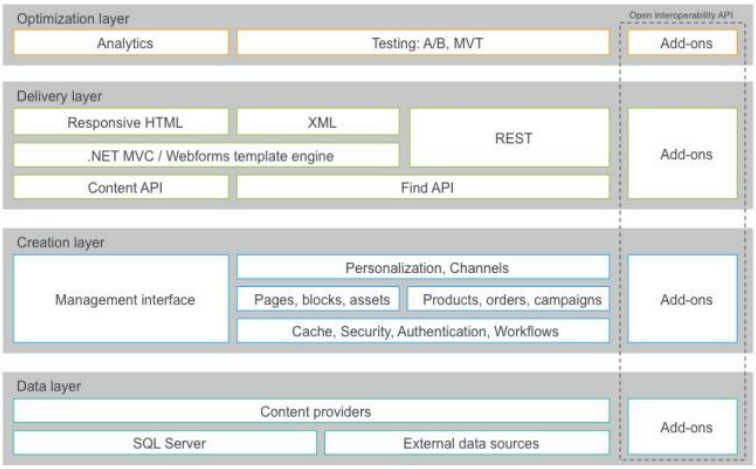
Episerver Content



Episerver

Episerver CMS System Review – Architecture and API overview

Episerver Technical overview



Episerver

Episerver Technical overview

- 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.
- 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 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 optimization layer provides various options for measuring, analyzing and optimizing the performance of website content, for instance conversion rates for a campaign landing page.

Episerver CMS System Review – Architecture and API overview

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

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

1. **ServiceLocator.Current.GetInstance<T>**: to 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);
}
```

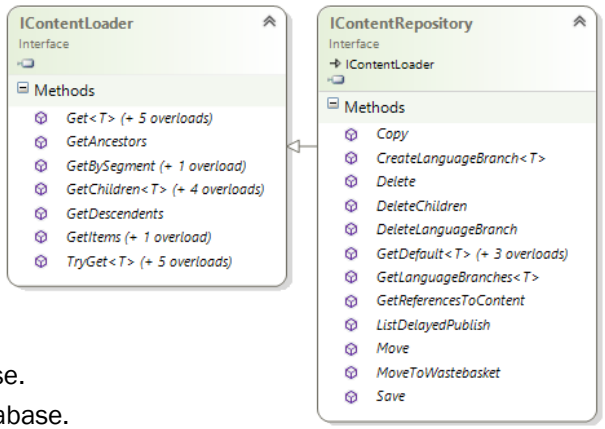
How to get an instance of a service with constructor dependency injection

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 DI container to be configured. The Alloy site has an example of how to do this.

API access to the Episerver CMS database



The **DataFactory** has grown too big should be avoided for performance and unit testing.

Interfaces and service locators should be used.

The two most common are:

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

Note: 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**. This is a form of technical debt due to the design decision to use the concept of “pages”.

Other Episerver interfaces and types for services

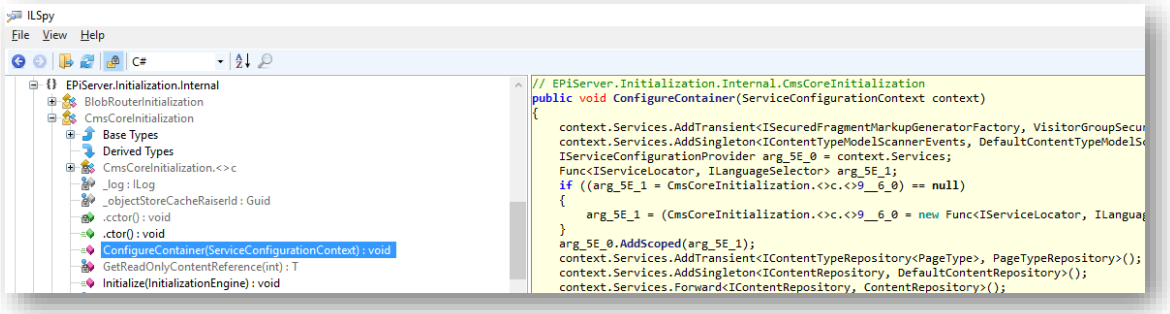
IContentTypeRepository: CRUD with content types in Episerver database.
IContentVersionRepository: list and delete content versions in Episerver database.
IContentEvents: listen for events during CMS lifecycle, e.g. publishing a page.
IPageCriteriaQueryService: search for content in Episerver database (not indexed).
UrlResolver: convert content reference to public URL and other tasks.
LocalizationService: read localized strings from XML files (or custom provider).
DisplayOptions: customize which template is used for a block.

Dependency injection

http://world.episerver.com/documentation/Items/Developers-Guide/Episerver-CMS/9/Initialization/dependency-injection/

Episerver CMS System Review – Architecture and API overview

Dependency injection for services




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

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 = context.Locate.Advanced.GetInstance<IContentVersionRepository>();
        }

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

Make the system use your component

Episerver

Episerver CMS System Review – Architecture and API overview

IContentRepository / IContentLoader

- Get the interface from IOC container:

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

IContentLoader loader = EPiServer.ServiceLocation.ServiceLocator.Current.GetInstance<IContentLoader>();
```

- Example, get a TextPage instance:

```
TextPage page = loader.Get<IContent>(pageLink) as TextPage;
```

No exception thrown if the page is not a TextPage

- Example, update and publish a page instance:

```
TextPage writablePage = repo.Get<TextPage>(pageLink).CreateWritableClone() as TextPage;

writablePage.MainBody = new XhtmlString("something");
repo.Save(writablePage, SaveAction.Publish);
```

TypeMismatchException will be thrown if the page is not a TextPage

Episerver

Episerver CMS System Review – Initialization

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/Items/Developers-Guide/Episerver-CMS/9/Initialization/Initialization/

Episerver CMS System Review – Initialization

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…

Episerver CMS System Review – Initialization

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;
        }
    }
```

Episerver

Episerver CMS System Review – Initialization

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;
    }
}
```

Episerver

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
  - `private static readonly ILogger Logger = LogManager.GetLogger();`
- 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

Episerver CMS System Review – Logging

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:

Episerver CMS System Review – Logging

Write to log example

- Get the logger

```
using EPiServer.Logging;
public partial class MyControl : PageControllerBase<PageData>
{
    private static readonly ILogger _logger = LogManager.GetLogger(typeof(MyControl));
```

- Write to the log

```
_logger.Warn("My custom warning message");
```

Episerver CMS System Review – Security

Security topics

- Roles and views
- Groups and Access right
- Security in the Episerver project templates
  - Security in CMS 10 Alloy site
  - OWIN Startup in CMS 10 Alloy site
- 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 CMS System Review – Security

Roles and views

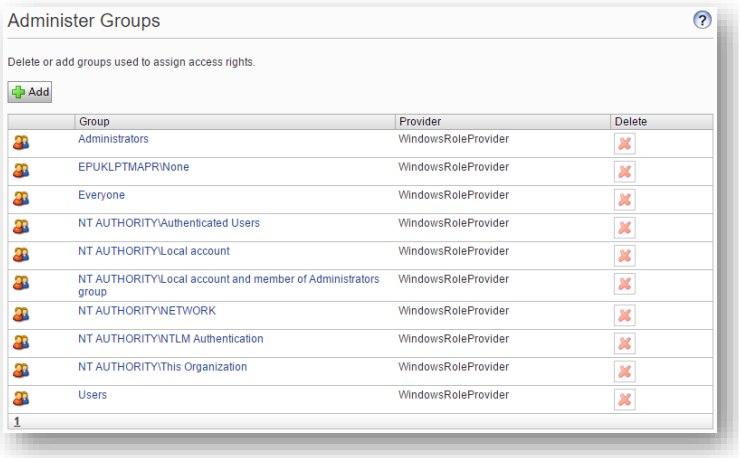
For the following views, who can access them, and what can they be used for?

View	Who can see the view?	What menus can they access?
Admin	Members of the virtual role **CmsAdmins**. By default, members of **WebAdmins** SQL group and **Administrators** Windows group are mapped to CmsAdmins in Web.config.	All, including CMS **Edit**, **Admin**, **Reports**, and **Visitor Groups**. However, by default only Administrators are given full permissions to content. The WebAdmins group must be given permissions to the Root manually.
Edit	Members of the virtual role **CmsEditors**. By default, members of **WebEditors** SQL group are mapped to CmsEditors in Web.config.	Only CMS **Edit** and **Reports**. However, by default they are given only read permission to content. The WebEditors group must be given more permissions to the Root.
Live	Members of the **Everyone** Windows group.	Read any content.

Groups

What is the difference between the following groups? What can they do? Do all of the groups exist automatically?

- Administrators
- WebAdmins
- WebEditors
- VisitorGroupAdmins
- Everyone

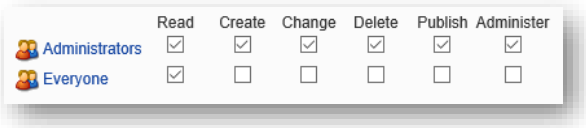


Episerver CMS System Review – Security

Access rights

What are the six access permissions?

- Read
- Create
- Change
- Delete
- Publish
- Administer



Does Administer rights give access to the Admin View?

- No

Security in the Episerver project templates

The CMS 9 project templates (and the CMS 10 Empty project template) configure 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/Items/Developers-Guide/Episerver-CMS/9/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. Add the user to one of the groups: **Administrators**, **WebAdmins**, or **WebEditors**.

Episerver CMS System Review – Security

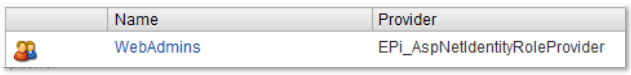
Security in CMS 10 Alloy site

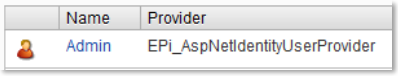
The Episerver CMS 10 Visual Studio extension project template for the Alloy sample site clears all the membership or role providers, and sets authentication mode to None. In combination with an OWIN startup class (next slide), this activates the **ASP.NET Identity** claims-based authentication system.

```
<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 CMS 10 site on the local server machine, it prompts you to create an **Admin** user.





Episerver CMS System Review – Security

OWIN Startup in CMS 10 Alloy 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
```


Episerver CMS System Review – Security

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.
- **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.

Episerver CMS System Review – Security

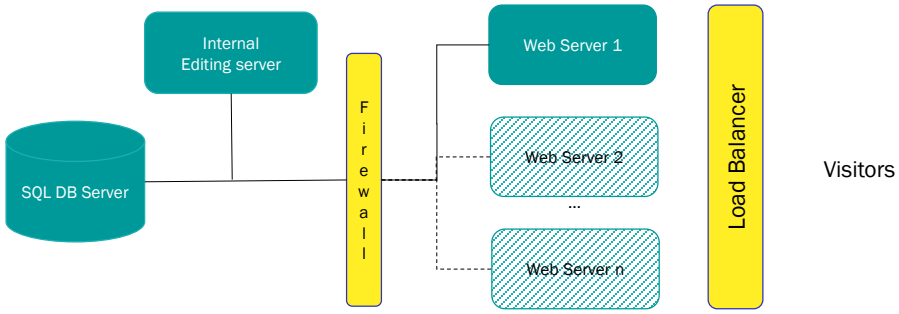
Common security mistakes

- **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.

Episerver CMS System Review – Security

Separate editing server

- The recommendation is to have the Edit and Admin UI on a separate server, only accessible from the internal network

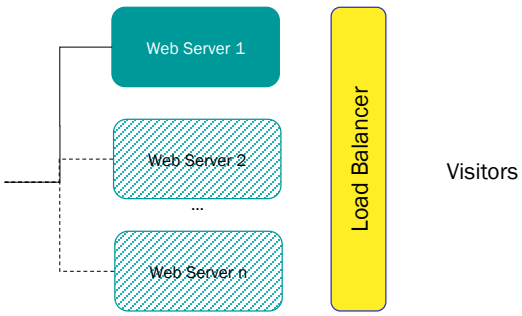


Episerver

Episerver CMS System Review – Security

Separate visitor servers

- On the public facing server (Web Server 1 (and web server 2– n if it was a load balanced scenario)):
  - Remove access to the edit and admin UI
  - Remove any custom edit- and admin plug-ins used for edit and admin
  - Remove the Windows membership- and role providers from the Multiplexing configuration if the site is not using it. It is unnecessary to expose the Windows user directory to anyone that can edit the site.



Episerver

Secure the UI folder

- If it is not possible to have the UI on a separate server:
  - In the IIS for the site:
    - Add a specially designated (and restricted) port for access to the uiUrl
    - Configure the uiUrl to use the restricted port
  - Use **SSL (Secure Sockets Layer)** to secure the website and/or UI folder

Episerver CMS System Review – Security

Restrict access to the Edit and Admin UI

- In web.config on the publicly facing load balanced visitor-only servers:
  - for **<location path="**<*UI*>**">** and
    <location path="<*UI*>**/CMS/admin">**,
    remove any allow roles:

```
<location path="newuipath/CMS/admin">
   <system.web>
     <authorization>
       <allow roles="WebEditors, WebAdmins, Administrators" />
       <deny users="*" />
     </authorization>
   </system.web>
</location>
```

Episerver CMS System Review – Security

Configure 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

Episerver CMS System Review – Security

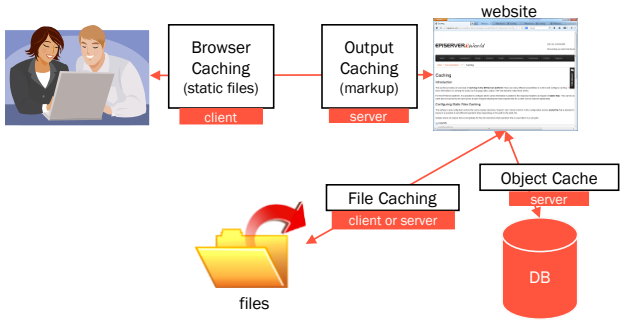
Remove configuration that is not used

- If using the Multiplexing membership- and role providers:
  - Remove the Windows provider from the list if not in use

```
<membership defaultProvider="MultiplexingMembershipProvider"
            userIsOnlineTimeWindow="10"
            hashAlgorithmType="HMACSHA512">
    <providers>
      <clear />
      <add name="MultiplexingMembershipProvider"
           type="Episerver.Security.MultiplexingMembershipProvider, Episerver.Framework"
           provider1="SqlServerMembershipProvider"
           provider2="WindowsMembershipProvider" />
```

Episerver CMS System Review – Caching

How Episerver caches content



Episerver

Episerver CMS System Review – 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.

Episerver

Episerver CMS System Review – Caching

Object cache

- Based on the ASP.NET Cache.
- Automatically caches all objects in Episerver CMS that is being requested from the API, via for example IContentRepository.
- Only read-only objects are stored to enable great performance.
- Invalidation is handled by the Remote Events system with support for load balanced servers.
- 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.

Episerver CMS System Review – Caching

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.

Episerver CMS System Review – Caching

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.

Episerver CMS System Review – Caching

Common questions on output caching

In the Episerver CMS SDK and Developer Guide, the recommendation is that output caching should 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?

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

Episerver CMS System Review – 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)
```

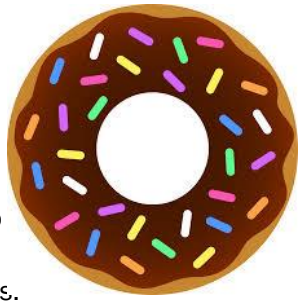
```
<episerver>
  <applicationSettings
    httpCacheability="Public"
    httpCacheExpiration="02:00:00"
```

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

Episerver CMS System Review – Caching



Output caching with ASP.NET Web Forms

Output caching stores the response sent is sent to the browser temporarily so subsequent requests the page doesn't need to re-execute.

In ASP.NET Web Forms, a developer could enable output caching in three ways.

- To cache an entire page or to cache part of a page (aka **donut *hole* caching**), at the top of a Web Form .aspx or a Web User Control .ascx:

```
<%@ OutputCache Duration="1200" %>
```

- To cache an entire page EXCEPT part of the page, aka **donut caching**, in the middle of markup:

```
<asp:Substitution runat="server" MethodName="GetTime" />
```

Unfortunately, although ASP.NET MVC has built-in support for entire page and partial page output caching, it does not have the equivalent of the Substitution control.

Episerver

Episerver CMS System Review – Caching

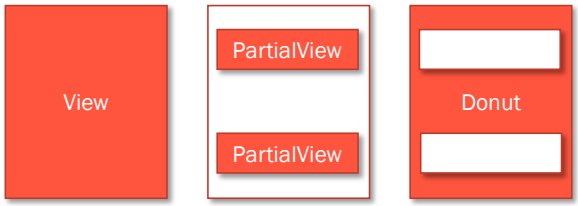
What types of output caching are supported in Episerver?

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

Episerver CMS System Review – Caching

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

Use Visual Studio’s Package Manager Console to install the NuGet package:

```
Install-Package MvcDonutCaching
```

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.

Episerver CMS System Review – 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));
```

Caching with Remote Events

- Cache is emptied when a page is updated
- Load balanced environment: all involved servers need to be updated at the same time, so use Remote Events (based on WCF) to do this
  - Can be done via UDP or TCP

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

Episerver CMS System Review – 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/

Episerver CMS System Review – Good practice

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
- Deploy in Release mode, not Debug, when project is released

Episerver CMS System Review – Good practice

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 many Episerver-versions on a web server means that more memory is needed
- Edit/Admin UI should be on a separate server not only for security but also for performance, especially if many editors are using it in parallel
- Commerce Manager should be on a separate server both from a performance- and a security perspective

Episerver CMS System Review – Good practice

More good practice for performance and security

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/

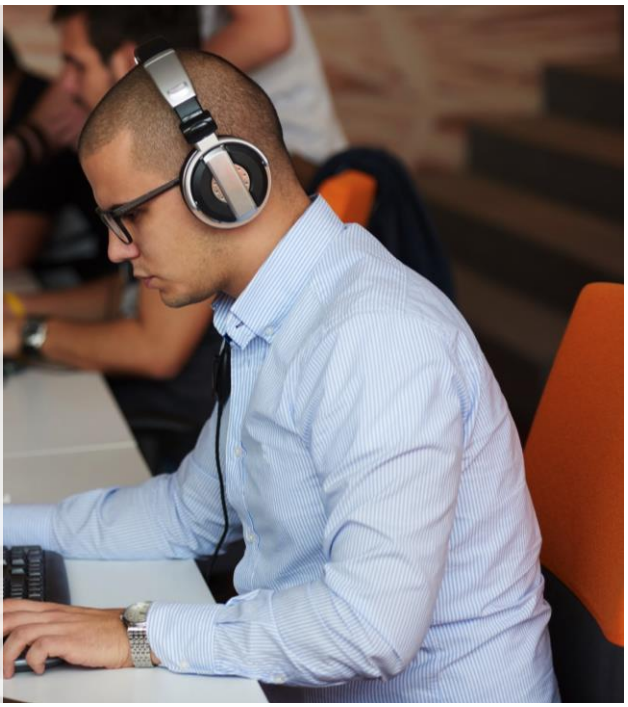
Recommendations for ASP.NET security settings:
http://world.episerver.com/documentation/Items/Developers-Guide/Episerver-CMS/9/Security/Recommendations-for-ASPNET-security-settings/

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

Episerver CMS System Review

Exercises for Module A

1. Implementing logging
2. Securing an Episerver site
3. Enabling beta features
4. Controlling the caching of responses
5. Listening for events and customizing services with initialization modules



Episerver

Episerver CMS Advanced Development

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.

Episerver

Working with Content using APIs

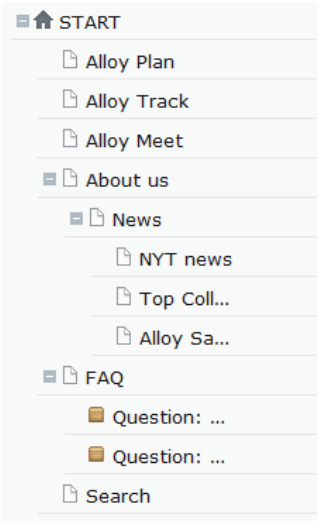
Topics

- The most important CMS classes
- Filtering
- Create, update and delete content
- Content versions and workflows
- Sharing information across content
- IContent concepts
- Managing access rights for content programmatically

Working with Content using APIs

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



Episerver

Controlling the page tree depth and children

Recommendation: less than 100 children in each page node.

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

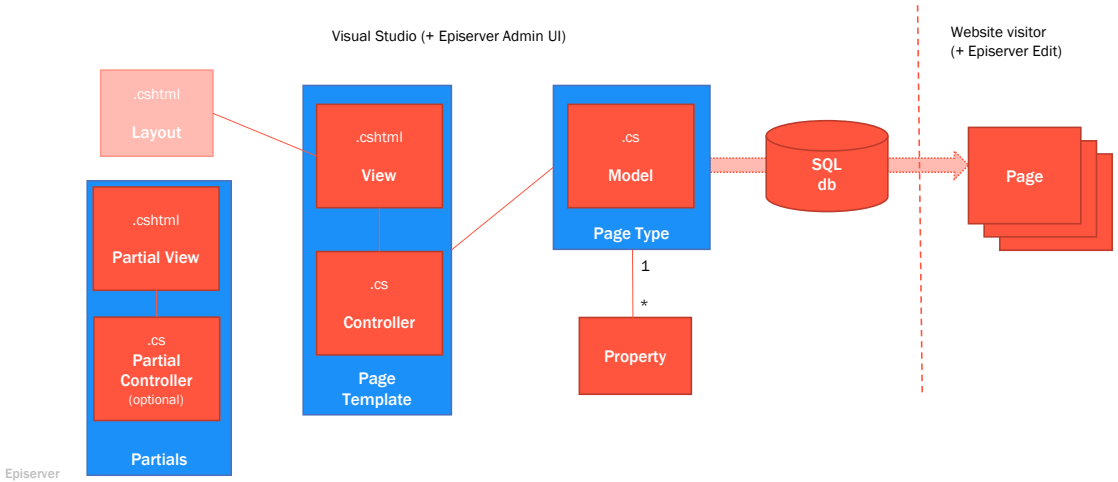
- The editor is creating news pages from the UI.
- The developer has added code for a “create news page” button and logic that will automatically place the page in the corresponding folder in the page tree: `/About Us/News/`
- This saves work for the editor as they do not need to find the correct place 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>`

Controlling the page tree page types

Exclude and ExcludeOn take priority over Include and IncludeOn when they conflict.

```
[AvailableContentTypes(
    Include = …,
    IncludeOn = …,
    Exclude = …,
    ExcludeOn = new Type[] { typeof(Alpha) },
    Availability = Availability.Specific)]
```

How it all comes together



Episerver

Working with Content using APIs

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.

Episerver

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.

How it all comes together – page templates

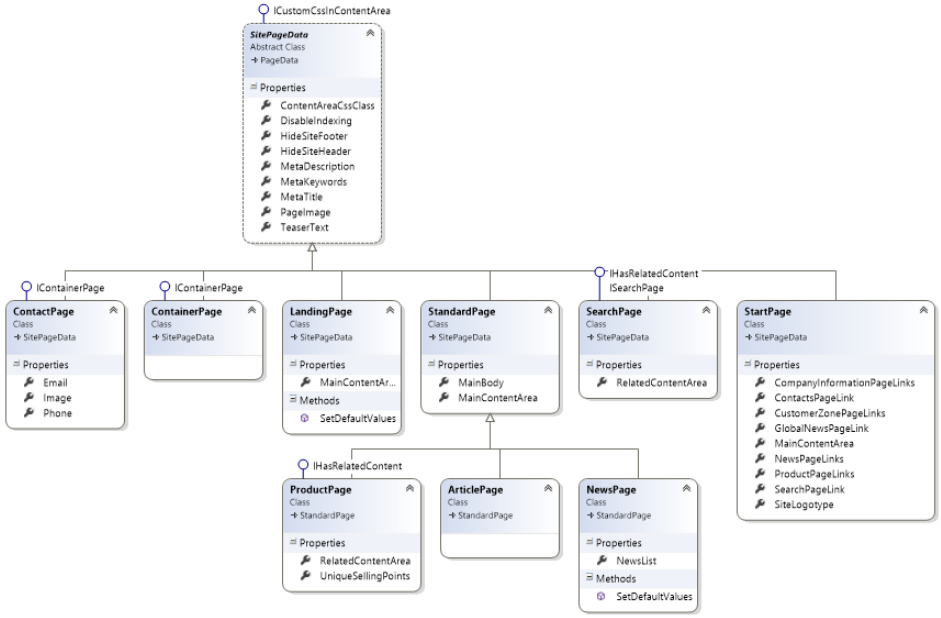
When a page is requested by a visitor, the most suitable page template that matches the context and is associated with the page's page type is used for displaying the content of the page.

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.

Alloy

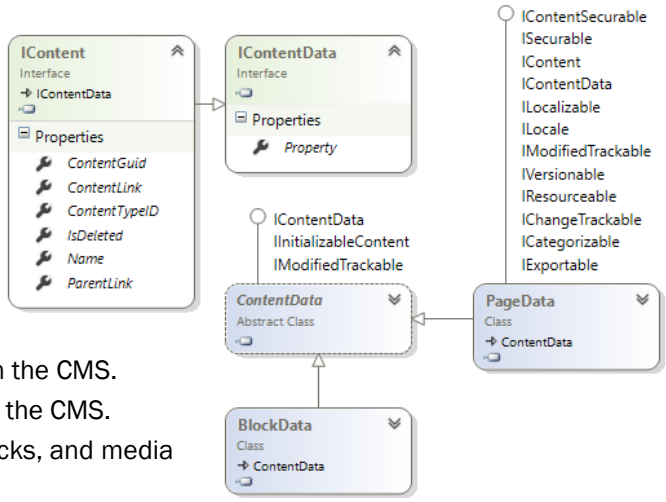
Add a **Class Diagram** to your project to document your page type inheritance hierarchy, for example, like this one for the Alloy web site:



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.

Page links and references

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

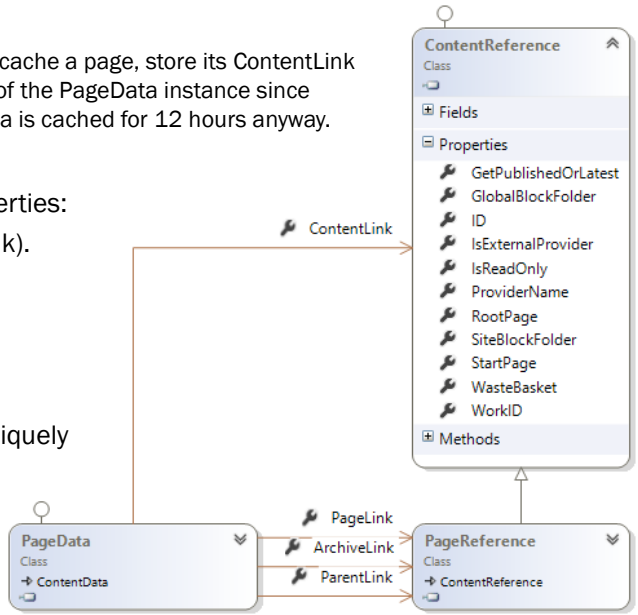
Every **PageData** instance has three "link" properties:

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

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

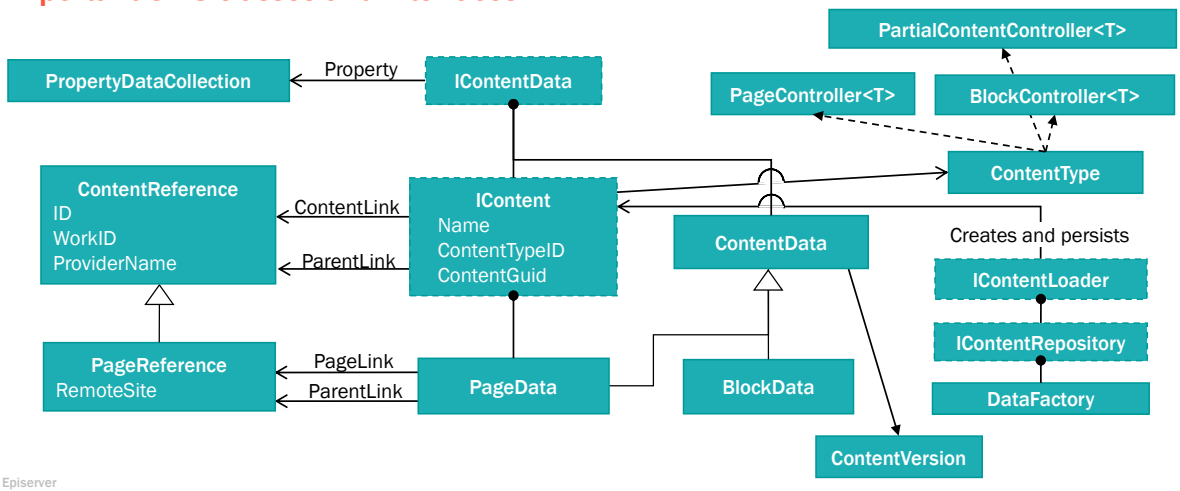
A **PageReference** is a **ContentReference** and a **ContentReference** has three properties that uniquely identifies a **ContentData** instance:

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



Episerver

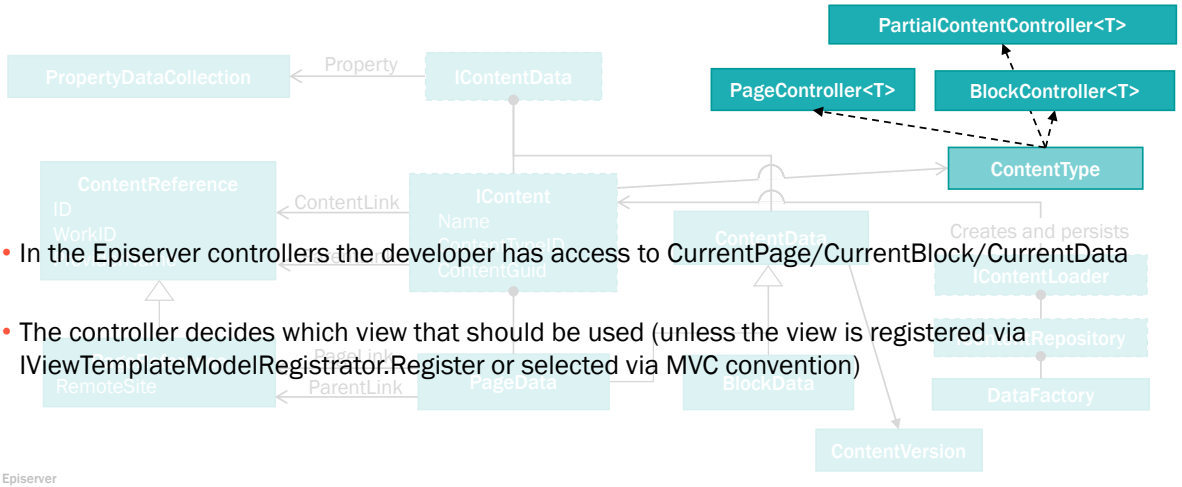
Important CMS classes and interfaces



Episerver

Working with Content using APIs – Important CMS classes and interfaces

Controllers and views



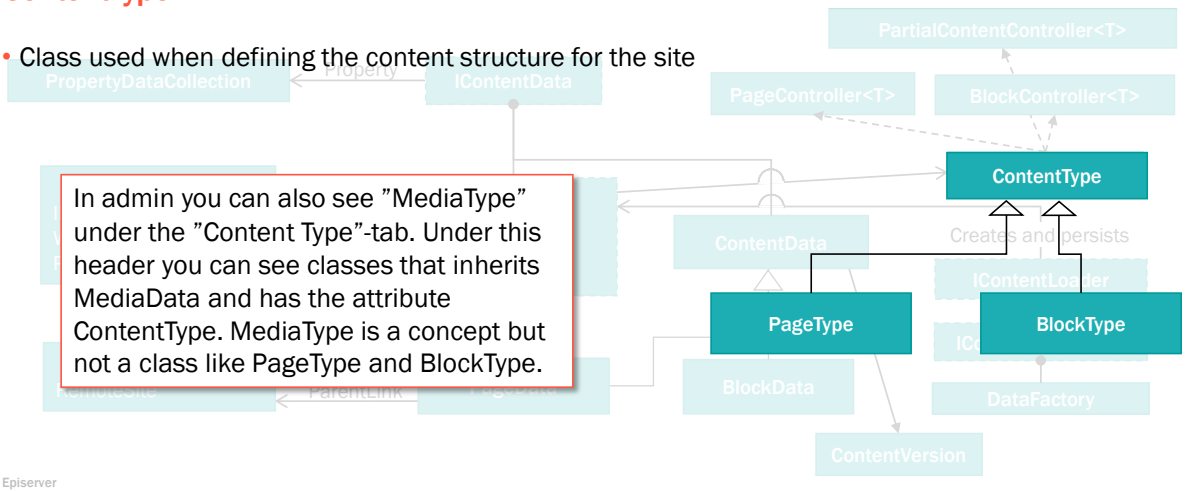
- In the Episerver controllers the developer has access to CurrentPage/CurrentBlock/CurrentData
- The controller decides which view that should be used (unless the view is registered via IViewTemplateModelRegistrator.Register or selected via MVC convention)

Episerver

Working with Content using APIs - Important CMS classes and interfaces

ContentType

- Class used when defining the content structure for the site



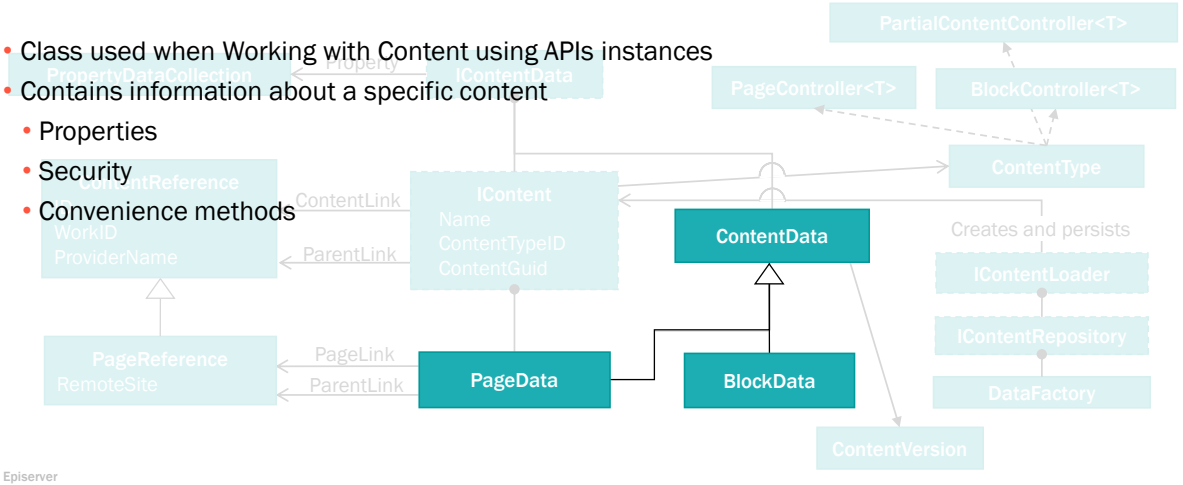
In admin you can also see ”MediaType” under the ”Content Type”-tab. Under this header you can see classes that inherits MediaData and has the attribute ContentType. MediaType is a concept but not a class like PageType and BlockType.

Episerver

Working with Content using APIs - Important CMS classes and interfaces

ContentData

- Class used when Working with Content using APIs instances
- Contains information about a specific content
  - Properties
  - Security
  - Convenience methods

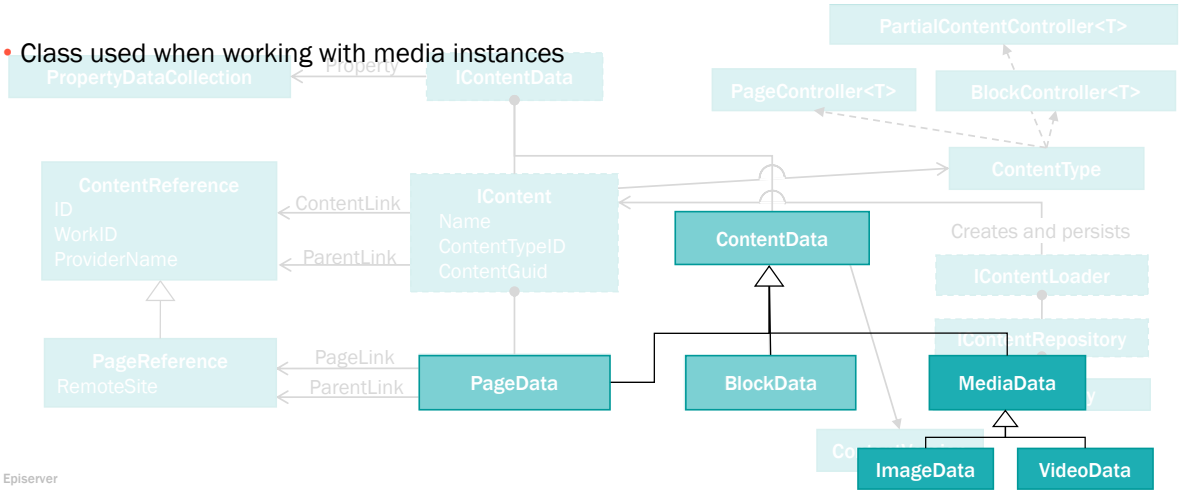


Episerver

Working with Content using APIs - Important CMS classes and interfaces

MediaData

- Class used when working with media instances

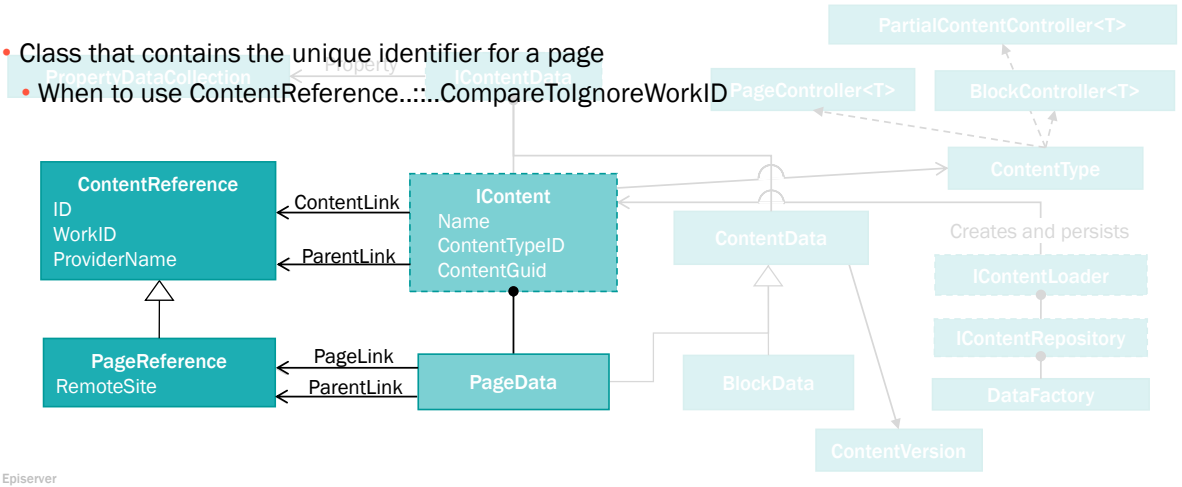


Episerver

Working with Content using APIs – Important CMS classes and interfaces

ContentReference

- Class that contains the unique identifier for a page
- When to use ContentReference..::..CompareToIgnoreWorkID



Episerver

Working with Content using APIs – Important CMS classes and interfaces

PageReference versus ContentReference

Used to point to a specific Episerver page (and/or version). A page can have several versions and even be located on another physical server.

The PageData class contains information about a specific page. This includes the name of the page (PageName), reference (PageLink) and URL (LinkURL = template name + page ID).

When the Content concept and the IContent interface was introduced in Episerver CMS 7, ContentReference became a base class to PageReference and the property RemoteSite was made obsolete and replaced with the better named ProviderName.

PageData now inherits the IContent interface and the properties on the PageData class previously called PageName, PageTypeID and PageGuid are now pointers to the more generic Name, ContentTypeID and ContentGuid in the IContent interface.

When working with a PageData object you can still use the old property names but beware that they are likely to be phased out going forward and the recommendation is to use the IContent interface.

Episerver

CompareToIgnoreWorkID

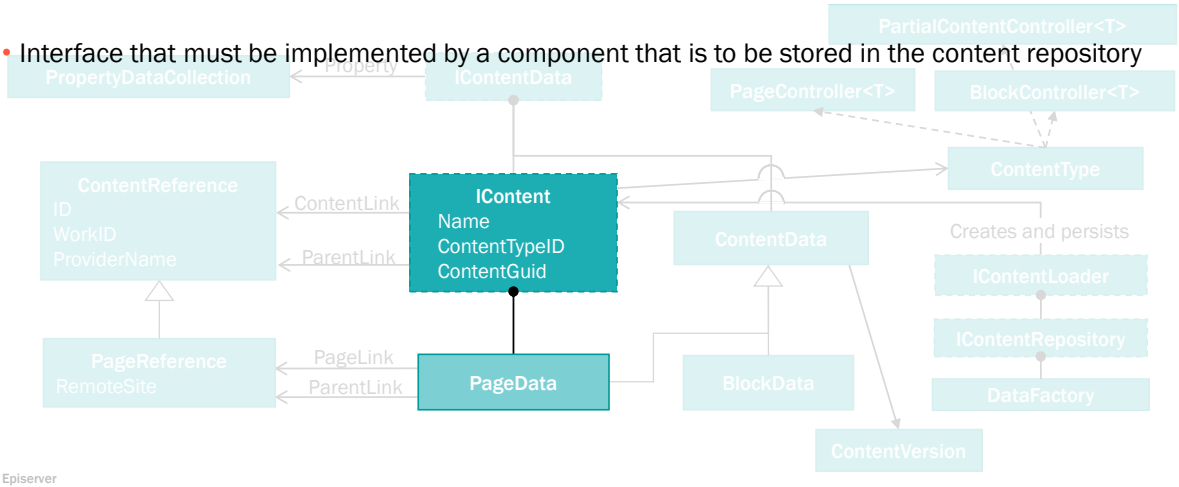
ContentReference.CompareToIgnoreWorkID compares two ContentReferences but ignores the WorkID (i.e. the content version).

Use this method when you for example build menus based on the page tree and are not interested in the different versions of each page that may exist.

Working with Content using APIs – Important CMS classes and interfaces

IContent

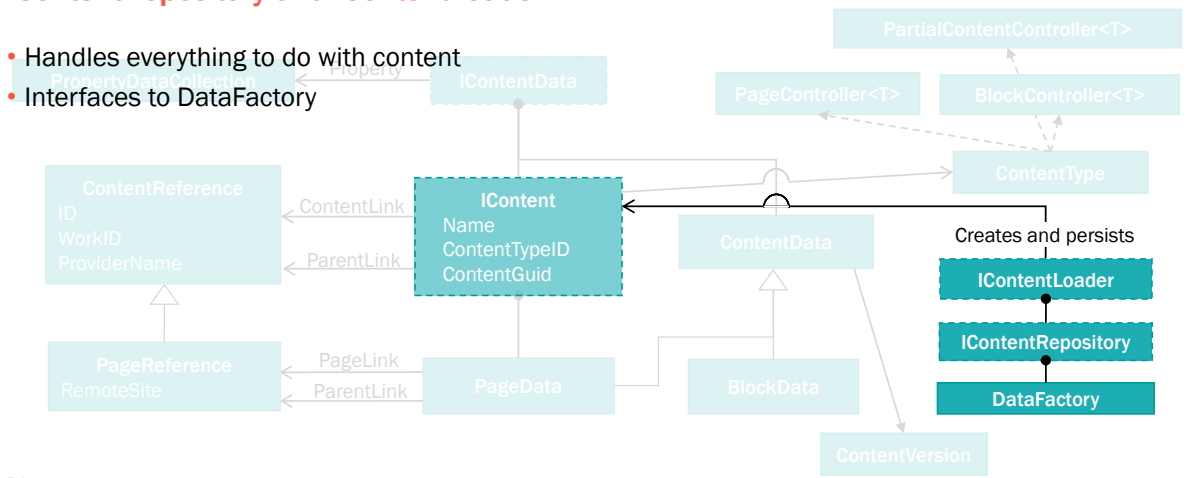
- Interface that must be implemented by a component that is to be stored in the content repository



Working with Content using APIs – Important CMS classes and interfaces

IContentRepository and IContentLoader

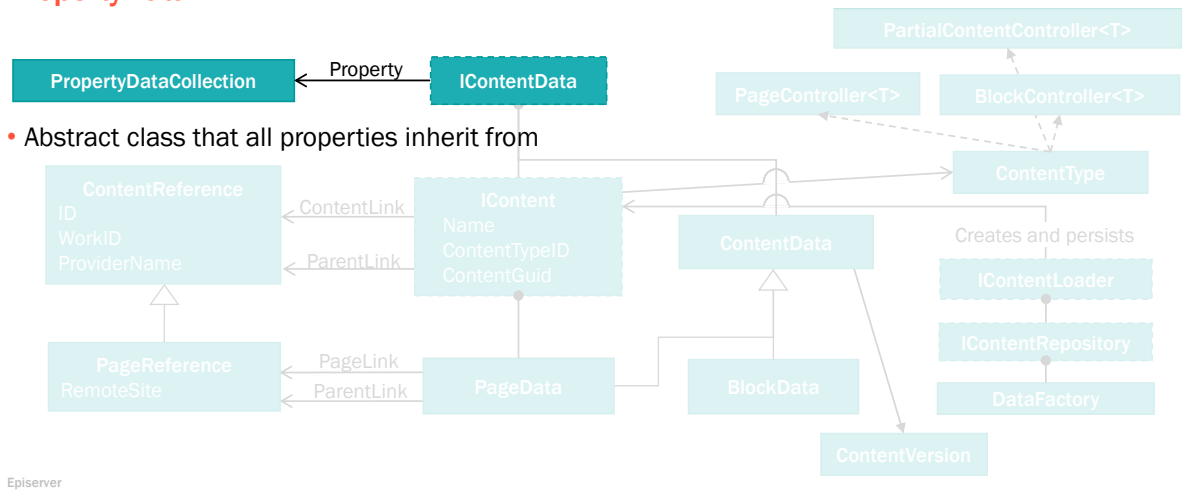
- Handles everything to do with content
- Interfaces to DataFactory



Episerver

Working with Content using APIs – Important CMS classes and interfaces

PropertyData



- Abstract class that all properties inherit from

Episerver

Working with Content using APIs – Important CMS classes and interfaces

Working with pages programmatically: Creating new page

- Decide where in the structure the page will live:

```
PageReference parent = PageReference.StartPage;
```

- Generate a new PageData object using IContentRepository:

```
IContentRepository contentRepository = ServiceLocator.Current.GetInstance<IContentRepository>();
NewsPage newsPage = contentRepository.GetDefault<NewsPage>(parent);
```

- Set required page properties:

```
newsPage.Name = "Today's news";
newsPage.NewsList.PageListRoot = PageReference.StartPage;
newsPage.MainBody = "<p>This is produced programmatically.</p>";
```

- Save the PageData object:

```
contentRepository.Save(newsPage, Episerver.DataAccess.SaveAction.Publish);
```


Working with Content using APIs – Important CMS classes and interfaces

Working with pages programmatically: Update an existing page

- Content instances are Read-Only by default
- Turn Read-Only into Read/Write by calling the CreateWritableClone method of the page object:

```
IContentRepository contentRepository = ServiceLocator.Current.GetInstance<IContentRepository>();
NewsPage existingNewsPage =
contentRepository.Get<NewsPage>(pageLink).CreateWritebleClone() as NewsPage;
```

- Set required page properties:

```
existingNewsPage.MainBody = "<p>This was updated programmatically.</p>";
```

- Save the PageData object:

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

Working with shared blocks programmatically

- Shared block needs to implement **IContent**
- Creating:

```
IContentRepository repository = ServiceLocator.Current.GetInstance<IContentRepository>();
var myNewBlock = repository.GetDefault<EditorialBlock>(ContentReference.GlobalBlockFolder);
myNewBlock.MainBody = new XhtmlString("<p>Hello World!</p>");

var content = myNewBlock as IContent;
content.Name = "MyNewSharedBlock";
ContentReference blockRef = contentRepository.Save(content, DataAccess.SaveAction.Publish);
```

- Updating:

```
EditorialBlock blockToUpdate = contentRepository.Get<EditorialBlock>(blockRef).CreateWritableClone() as EditorialBlock;
blockToUpdate.MainBody = new XhtmlString("<p>Hello again!</p>");
contentRepository.Save(blockToUpdate as IContent, DataAccess.SaveAction.Publish);
```


Working with Content using APIs – Important CMS classes and interfaces

CMS 10 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).

- **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.
- **CheckIn** - Checks in a version indicating that it is ready to be published
- **Publish** - Publishes a version. The currently published version will automatically transition to a previously published state.
- **Schedule** – Used to schedule a version for automatic publishing at a later date. (New in CMS 10)

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

Content versions

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, CheckedOut, AwaitingApproval, Rejected, CheckedIn, DelayedPublished, Published, PreviouslyPublished.

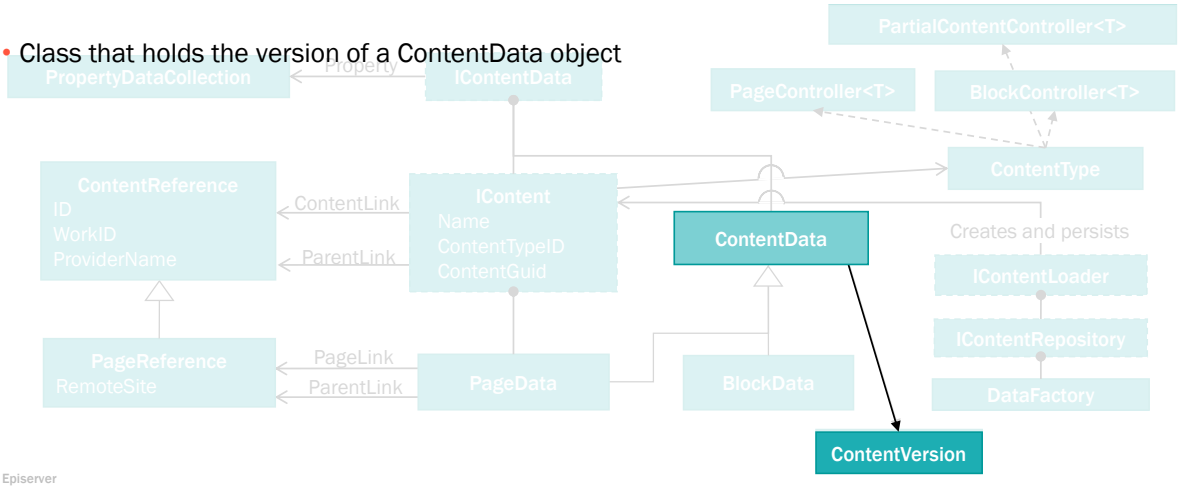
A state transition can programmatically be specified by parameter SaveAction in the call to IContentRepository.Save:

- Default/None, Publish, Schedule, CheckOut, CheckIn, RequestApproval, Reject, Save (deprecated).

Working with Content using APIs – Important CMS classes and interfaces

ContentVersion class

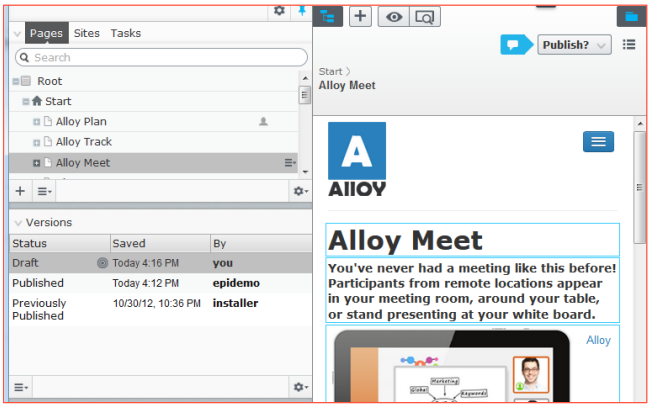
- Class that holds the version of a ContentData object



Working with Content using APIs – Important CMS classes and interfaces

Content versions

- The editor can see versions in Edit View by adding the Versions gadget



Episerver

Working with Content using 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)
```

Episerver

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.
IPageCriteria QueryService	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.
SearchClient	Find Index	Advanced capabilities, but extra licence.

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>

Filtering content

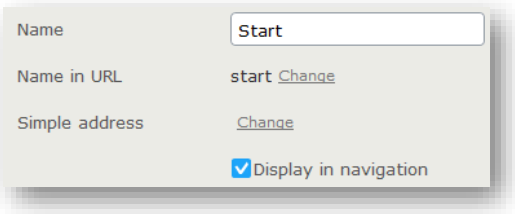
Type	Method	Parameter(s)	Return Type
FilterForVisitor	Filter	IEnumerable<IContent>	IEnumerable<IContent>
		PageDataCollection	PageDataCollection
FilterContentForVisitor	Filter	IList<IContent>	void*

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
    .OfType<PageData>().Where(p => p.VisibleInMenu);
```

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

Filter and sort content

- FilterForVisitor
  - Used to remove all content current user is not allowed to view
  - It also removes all content that is unable to be rendered in itself. Container pages, blocks and other content with partial or none renderers will be removed.
- FilterContentForVisitor
  - Lets you specify which renderers that should be allowed
    - WebFormsPartial
    - MvcPartialView
- Sort content
  - The old FilterSort only supports pages.
  - Use Linq queries to perform sorting.

Use on content that is rendered in itself

Use to customize the filter, for example allow partial templates.

Episerver.Framework.Web.TemplateTypeCategories

FilterSort accepts IContent but can only sort pages. Use Linq query instead.

Episerver

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

Using lambda expressions in LINQ

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

var pages = contentRepository.GetChildren<ProductPage>(ContentReference.StartPage);



var filteredPages = FilterForVisitor.Filter(pages).Take(3).Cast<ProductPage>();

var sortedPages = filteredPages.OrderBy(page => page.StartPublish);
```

We cast since FilterForVisitor.Filter returns IEnumerable<IContent>

For example, sort by StartPublish property in ascending order

Episerver

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)

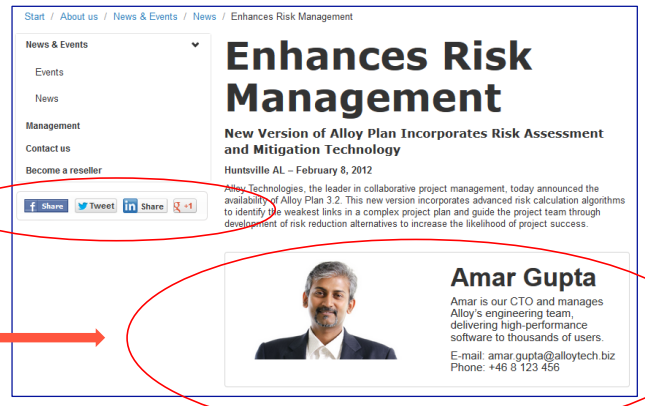
```
private readonly IPageCriteriaQueryService finder;
```

Type	Method	Parameter(s)	Return Type
IPageCriteria QueryService	FindPagesWithCriteria	PropertyCriteriaCollection	PageDataCollection

```
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);
```

Sharing information across content



"Share this" block

Block with contact person for this business area. Automatically retrieved based on news article type.

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 resides

```
var pageRouteHelper = EPiServer.ServiceLocation.ServiceLocator.Current.GetInstance<EPiServer.Web.Routing.PageRouteHelper>();

PageData myPageData = pageRouteHelper.Page;
```

- Scenario 2: Parent content that will automatically add the blocks it needs
  - Example:
    - A product page where blocks that contain related information are automatically added by default
  - Solutions:
    - Add a shared block to content area programmatically
    - Add blocks without having a content area, using the RendererContentData helper method

```
See code examples on the next slide(s)
```

Working with Content using APIs

Sharing information across content: Adding shared block to content area

- 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;
DataFactory.Instance.Save(page, SaveAction.Publish, AccessLevel.Publish);
```


Working with Content using APIs

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.

Some of the most commonly used 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

Working with Content using APIs

Example of working with Category

```
using Episerver.DataAbstraction;
```

finding

```
Category dataCat = Category.Find("Data");
if (dataCat != null)
{
          Response.Write("Data category was found and had ID:" + dataCat.ID.ToString());
          Category pcCat = dataCat.FindChild("PC");
          if (pcCat != null) Response.Write("Data has a child category PC");
}
```

creating

```
Category root = Category.GetRoot();
Category newCat = new Category("NewCategory", "My new test category");
root.Categories.Add(newCat);
newCat.Save();
```

and deleting

```
Category createdCat = Category.Find("NewCategory");
createdCat.Delete();
```

Working with Content using APIs

Essential 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**

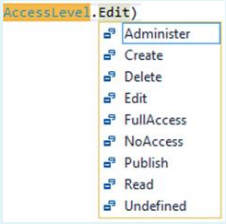
- When you create a shared block programmatically: why does it need to be cast to IContent before it can be saved?
  - Blocks can be used as property types as well as shared. Only shared blocks need to implement IContent.

Episerver

Working with Content using 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 {}
```

Episerver

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();
```

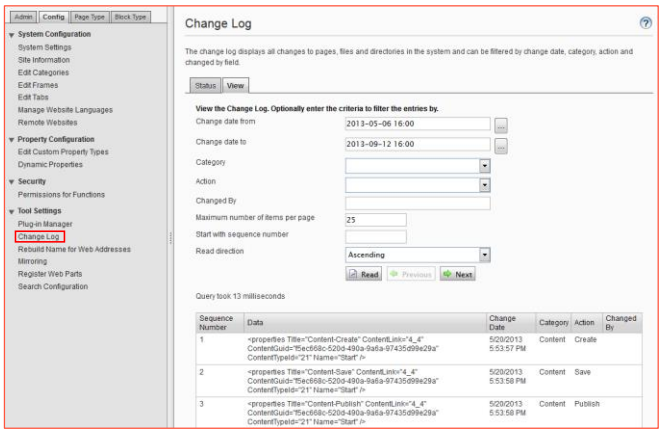
- Get notified when a content's ACL has changed:

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


Working with Content using APIs

Read the Change Log

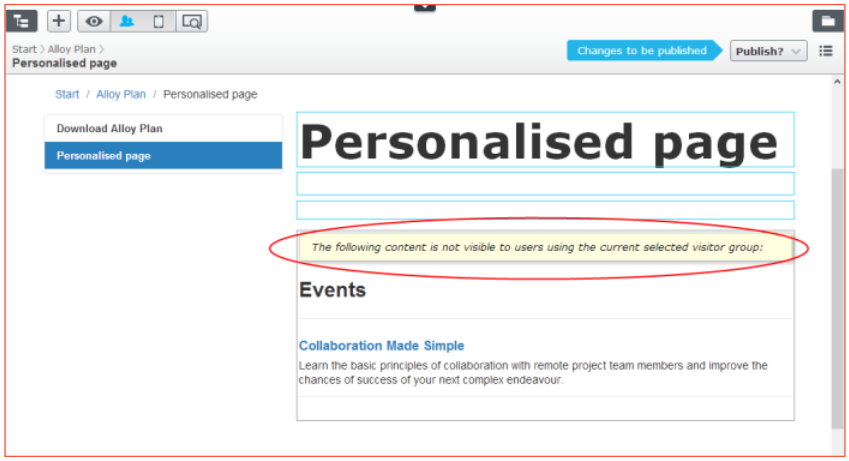
- In Admin:



- Read/write to it programmatically using the Episerver.ChangeLog namespace

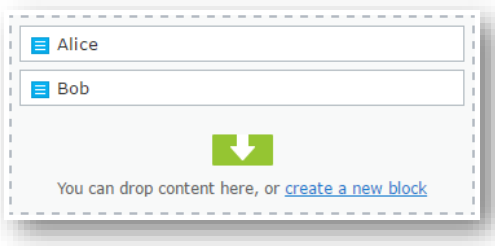
Working with Content using APIs

PageEditing.PageIsInEditMode



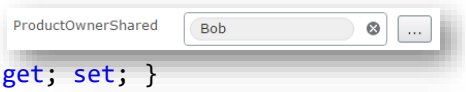
Working with Content using APIs – Content areas and blocks

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; }
```

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();
```


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

How to render a ContentReference

```
using EPiServer.ServiceLocation;
```

To render a ContentReference property in a view, you must consider what type of content it is:

- If it points to a page, do you want to render a **link** to that page?

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

- If it points to a block or a page, do you want to render the page using a **partial page template**?

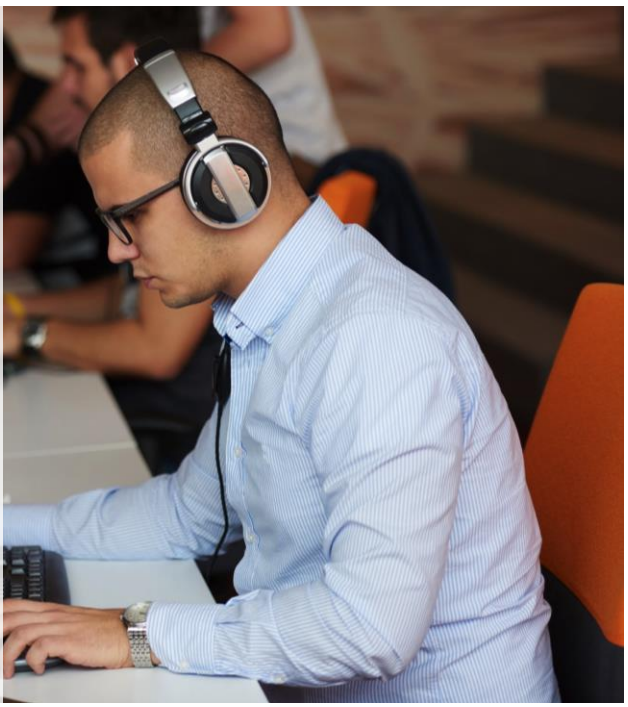
```
@{
    var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
    var content = loader.Get<IContentData>(Model.MyContentReference);
    Html.RenderContentData(content, isContentInContentArea: false);
}
```

If you have a block used as a property type, or a ContentArea with blocks, you can just use PropertyFor:

- `@Html.PropertyFor(m => m.MyBlockOnThePage)`

Exercises for Module B

1. Designing a commenting solution
2. Reviewing a potential solution
3. Implementing a Share This Page block
4. Completing some challenges



Episerver CMS Advanced Development

Customizing Properties

Episerver Properties are central in Episerver and something that the editor uses daily. Common editor tasks can often be solved through using a Episerver property, given that you as a developer know how the Episerver property works and how it can be modified.
This section is about the secrets behind the Episerver Property.

Customizing Properties

Topics

- Concepts
  - Important areas to build a powerful and customized website
- Episerver Property
  - Reveal the magic
- UI Hint EditorDescriptor Tag
  - What and when?
- ContentAreas and blocks
- Dojo

Customizing Properties – Concepts

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

Customizing Properties – Concepts

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	~~AllowedTypes~~
PropertyNumber	Byte	
	Int32	Range
PropertyPageType	PageType	
PropertyLongString	String	Textarea StringLength
PropertyTimeSpan	TimeSpan	
PropertyUrl	Url	Image Document Video
PropertyXForm	XForm	
PropertyXhtmlString	XHtmlString	
PropertyBlob	Blob	
PropertyContentReferenceList	IList<ContentReference>	AllowedTypes

Customizing Properties – Concepts

Property type example

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

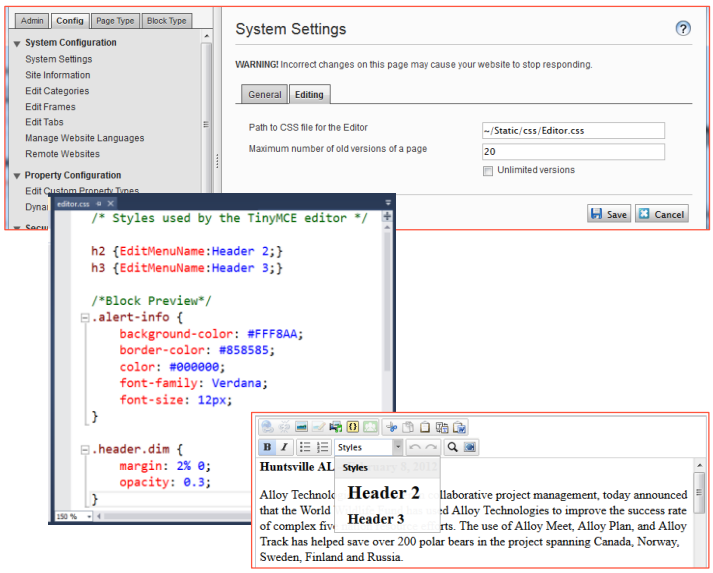
```
[Display(GroupName = SystemTabNames.Content, Order = 310)]
[CultureSpecific]
public virtual XhtmlString MainBody { get; set; }
```

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

Customizing Properties – Concepts

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.

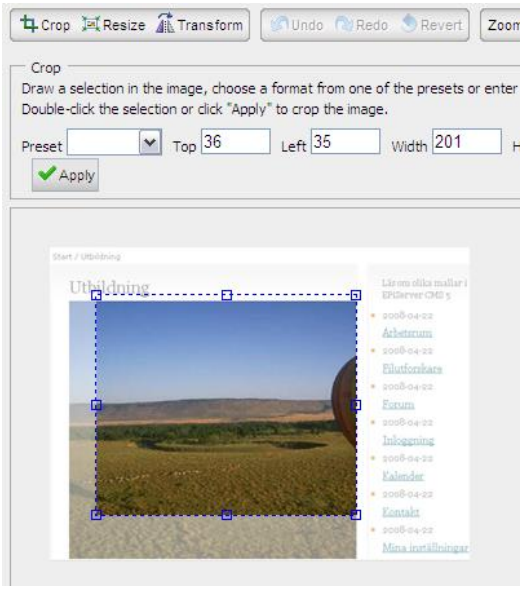


TinyMCE – Customizing the image editor

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" />
```



Customizing Properties – Concepts

Content type common properties

Although the **PageData** class does NOT define them, an Episerver *convention* is to give pages the following properties:

- **MainIntro**: a string for a short introduction to the page (often used as fallback for MetaDescription).
- **MainBody**: an XhtmlString for the main rich content property.

Most developers familiar with Episerver will expect these two properties to exist so you should create them. It would also be good practice to define properties for the <head> in a base page type:

- **MetaTitle**, **MetaDescription**, **MetaKeywords**: string
- **MetaViewPort**: string e.g. “width=device-width, initial-scale=1.0”
- **LinkCanonical**, **LinkAlternates**: bool e.g. when true, call @Html.CanonicalLink(Model.ContentLink) in <head> section

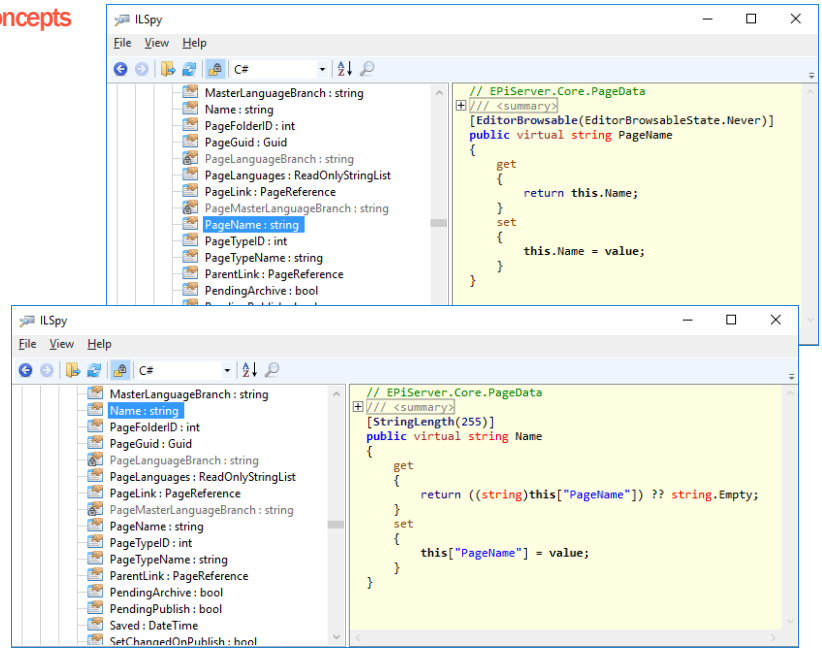
Customizing Properties – Concepts

Name or PageName?

Use tools such as *ILSpy* to find out what's actually happening internally in Episerver APIs.

Note

You don't see the PageName property in IntelliSense because it has been marked as [EditorBrowseableState.Never]. However, it is still there and compileable. We just do that to discourage its use. Use Name instead.

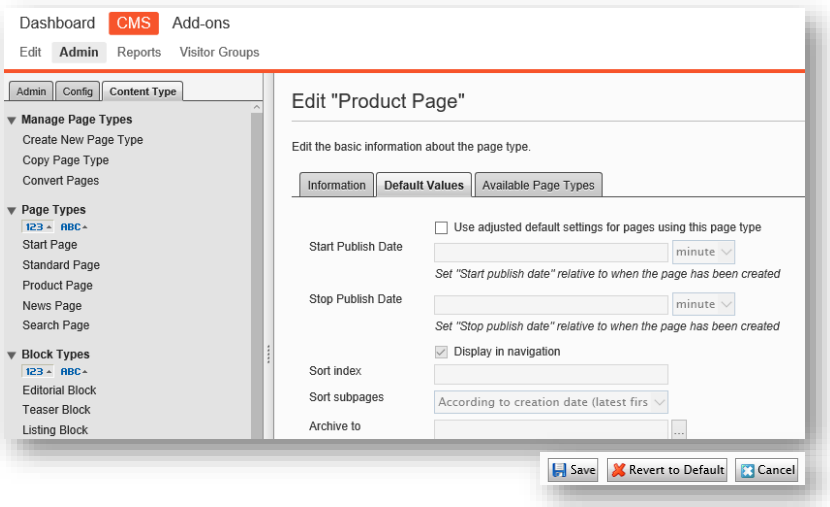


Episerver

Customizing Properties – Concepts

Setting default values

Default values can be set in code by overriding the method named **SetDefaultValues**, and some built-in properties can be set in Admin view.



Episerver

Customizing Properties – Type choices

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; }
```

Episerver

Customizing Properties – Type choices

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.	

Episerver

Customizing Properties – Type choices

Property types using Episerver types

Episerver Type	Purpose	Examples of common attributes
EPiServer.Core.**XhtmlString**	Rich text and blocks.	
EPiServer.**Url**	A link to a page, email, or external URL.	
EPiServer.SpecializedProperties.**LinkItemCollection**	A manually managed collection of links to pages, emails, or URLs.	
EPiServer.Core.**ContentArea**	An ordered collection of blocks and pages (rendered using their partial template).	[AllowedTypes(...)]
EPiServer.XForms.**XForm**	Enables selection and management of XForms.	

Customizing Properties – Type choices

Property types using Episerver reference types

Episerver Type	Purpose	Examples of common attributes
EPiServer.Core.**ContentReference**	A reference to one item of content.	[AllowedTypes(typeof(BlockData), RestrictedTypes = new Type[] { typeof(EventBlock) })] [UIHint(UIHint.Image/Video/MediaFile)]
EPiServer.Core.**PageReference**	A reference to one page.	[AllowedTypes(typeof(NewsPage), typeof(ArticlePage))]
IList<ContentReference> *	A list of references to content. Note: you cannot have an IList<PageReference>!	[AllowedTypes(...)]

* The official documentation shows **ContentReferenceList**. This does NOT exist! Use **IList<ContentReference>**.

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

Customizing Properties – Type choices

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*

Customizing Properties – Type choices

Collections of links

The **LinkItemCollection** is good choice for a property if you want to allow editors to *manually* control a collection of links to: pages, assets, external pages, e-mails. But if the editor creates a manual link to a page, it could cause a 404 if the page is removed or expires. Although the Alloy sample site uses this technique, it's often not a good choice.

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 (with ContentReference) 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.

Customizing Properties – References

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.
- **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.

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

How to render a ContentReference

```
using EPiServer.ServiceLocation;
```

To render a ContentReference property in a view, you must consider what type of content it is:

- If it points to a page, do you want to render a **link** to that page?

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

- If it points to a block or a page, do you want to render the page using a **partial page template**?

```
@{
    var loader = ServiceLocator.Current.GetInstance<IContentLoader>();
    var content = loader.Get<IContentData>(Model.MyContentReference);
    Html.RenderContentData(content, isContentInContentArea: false);
}
```

If you have a block used as a property type, you can just use PropertyFor:

- `@Html.PropertyFor(m => m.MyBlockOnThePage)`

Customizing Properties – 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; }
```

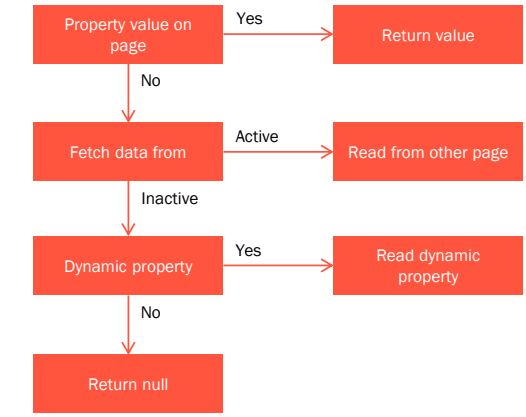

Customizing Properties – Validation

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
    }
```

Customizing Properties – 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.

Episerver

Customizing Properties – Rendering

Add property to template

- Use **PropertyFor** to get onPageEdit in matching property editor
  - Handles property value for visitors and editors
- Use **EditAttributes** to enable onPageEdit without PropertyFor

```
<h1>@Html.PropertyFor(x => x.CurrentPage.Name)</h1>

                          ==

<h1 @Html.EditAttributes(x => x.CurrentPage.Name)>
   @Model.CurrentPage.Name
</h1>
```

Episerver

ASP.NET MVC View helper methods

The following are *Microsoft* HtmlHelper methods:

- **Raw**: disables the automatic @ HTML encoding.
- **Display**, **DisplayFor**: outputs the value of the property and reflects over the model class for hints about which display template to render or what display format to use. ~/Views/Shared/DisplayTemplates
- **DisplayNameFor**: outputs the name of the property i.e. used for a label or reads the [Display] or [DisplayName] attribute on the model class.
- **Editor**, **EditorFor**: reflects over the model class for hints about which editor template to render. ~/Views/Shared/EditorTemplates
- **Partial**, **RenderPartial**: outputs the specified partial view (.cshtml) into the current view.

Customizing Properties – Rendering

ASP.NET MVC View helper methods containing "Action"

What is the difference between the following *Microsoft* HtmlHelper methods?

Method	Description
Html.ActionLink	Outputs a hyperlink using the Default MVC route e.g. /{controller}/{action}
Ajax.ActionLink	Outputs a hyperlink using the Default MVC route e.g. /{controller}/{action} that makes an AJAX request to perform a partial page update.
Url.Action	Outputs a URL using the Default MVC route e.g. /{controller}/{action} for use with an element with a src attribute e.g.
Html.Action	Executes a controller's action method and returns the response as a string.
Html.RenderAction	Executes a controller's action method and outputs the response to buffer.

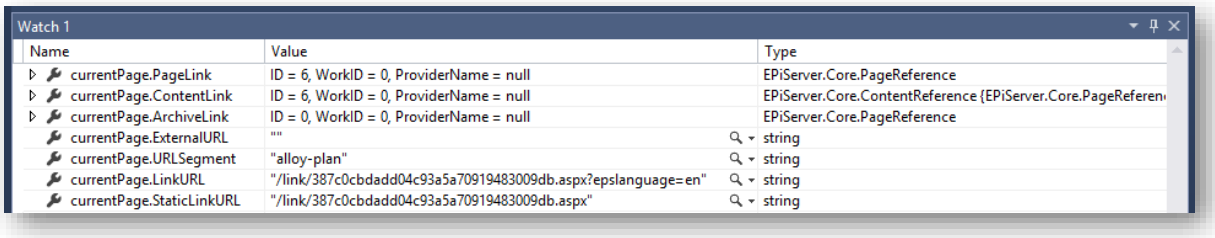
Note: none of these understand Episerver routes so they are rarely used.

Customizing Properties – Rendering

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



Episerver

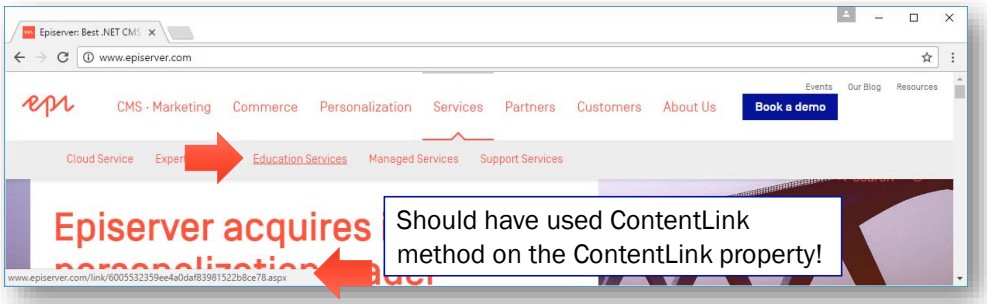
Customizing Properties – Rendering

@Html.ContentLink(Model.CurrentPage.ContentLink)

Episerver View helper methods

Episerver HtmlHelper extension methods:

- **PropertyFor**: renders a property with on-page edit support.
- **EditAttributes***: adds attributes to an element with on-page edit support.
- **ContentLink(*ContentReference*)**: renders a hyperlink <a href=""> for the specified content reference using its friendly URL segments.



Episerver

Customizing Properties – UIHints and Edit descriptors

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

Customizing Properties – UIHints and Edit descriptors

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

Customizing Properties – UIHints and Edit descriptors

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:

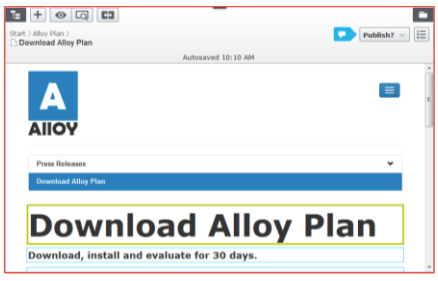
```
Episerver.Shell.UiWrapperType

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

Episerver

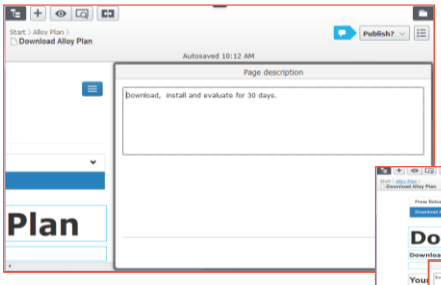
Customizing Properties – UIHints and Edit descriptors

UiWrapperType

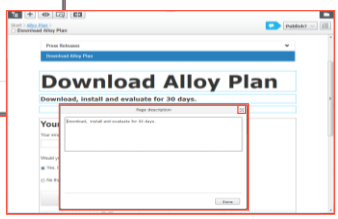
`UiWrapperType.ContentEditable`



`UiWrapperType.Flyout`



`UiWrapperType.Floating`



Episerver

Customizing Properties – Content areas

Customize ContentArea

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

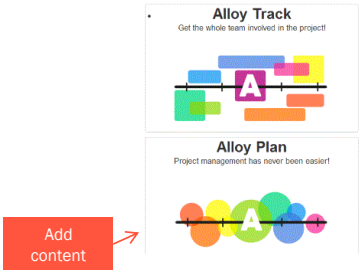
We are used to customizing the blocks and pages that are displayed by the content area.

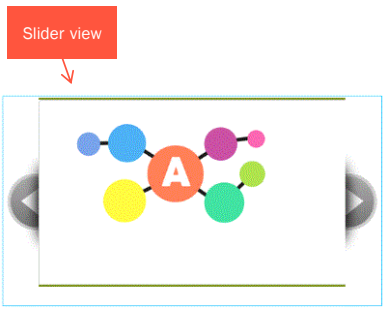
It is also possible to customize the content area itself

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/





Customizing Properties – Dojo

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.

Customizing Properties – Dojo

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.

Customizing Properties – Dojo

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);
            }
    });
});
```

Customizing Properties – Dojo

Dojo – Knowledge

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

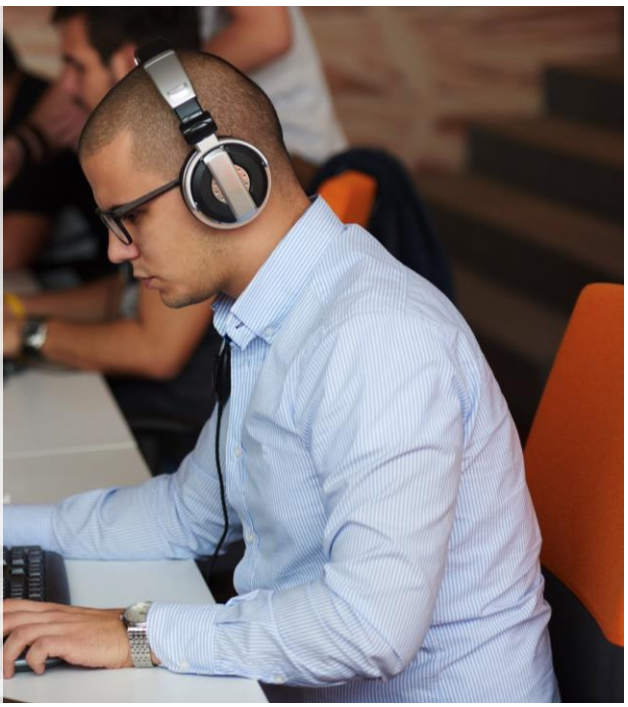
- References:
- http://world.episerver.com/Blogs/Linus-Ekstrom/Dates/2012/7/Creating-a-custom-editor-for-a-property/
- http://epiwiki.se/developing/plugins/dojo-widgets/simple-dojo-widget
- Also see the StringList editor in the Alloy sample site.

Episerver

Customizing Properties

Exercises for Module C

1. Implementing fall back properties
2. Moving properties to the basic info area
3. Selecting choices for property values
4. Review a custom property editor
5. Customize any property at runtime
6. Review a custom property type
7. Create a custom editing experience with Dojo



Episerver

Episerver CMS Advanced Development

Integrating Data

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. To not clutter the editorial interface with this data, you can instead choose to save it to the Dynamic Data Store, or you may need to integrate an external data store.

Episerver

Integrating Data

Topics

- Integration choices
- Partial routers
- Introduction to DDS
- What to store and why

Episerver

Integrating Data

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 and Swedish,
- Have access rights applied, and
- Have versions and publishing workflow.

Some of the data used by the Episerver CMS does not need those features, or it is stored outside the CMS database. But we might still want it to appear as part of a single system.

Integrating Data

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
Content Channel	One-way, read-only	Scheduled	Import data from an external system into content in the CMS. Episerver has written a *content channel* for SharePoint.	Medium
Scheduled Job	Two-way, read-write	Scheduled	Custom import/export data to/from an external system.	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

Integrating Data

Content Providers

For content that reside in an external repository and should behave as editorial content inside Episerver. For example:

- DB -> CMS: An external database with contacts where you want every contact to behave as a shared block in your CMS site.
- CMS -> CMS: Global company with several Episerver sites where all sites can have local news but want to share global news. The Global news are ”cloned” out to the other sites.

Information about how to implement and configure content providers is available in the ”Content Providers” section in the CMS SDK.

Integrating Data

Content Channels

For content that comes from an external source and is saved as content in Episerver.

The content is pushed to Episerver from the external system. (fire-and-forget)

Examples:

- Import product data and create a page for each, including images saved to the file manager
- Currency rate feed
- Episerver Connect for SharePoint is using content channels to push data to Episerver CMS

Integrating Data

Custom integration with Scheduled Jobs

Instead of using the built-in integration methods for content you can of course build your own custom integration using the capabilities in the .NET Framework.

Example:

- Scheduled job that reads data and inserts it in the CMS

You will learn more about scheduled jobs in Module H

Integrating Data

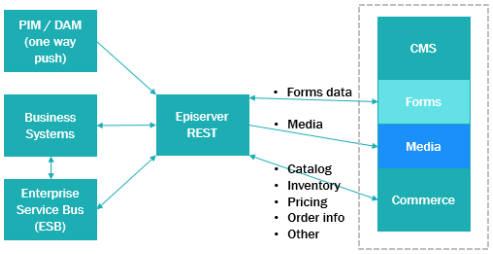
Scheduled job vs content provider

- Scheduled job vs content provider.
  - Content provider is often used when content is located at an external source but should be displayed in CMS/Commerce
  - Scheduled job retrieves data into EPi and creates content
- Which option you use depends on the requirements you have
- Scheduled job has been preferred because it is easier to work with than content providers:
  - The developer has more control
  - Less development is needed in comparison

Integrating Data

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.



The Service API provides a programming interface for performing operations like:

- Import and export of "episerverdata" files.
- Import and export forms data.
- Bulk import and export of 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.

Episerver

Integrating Data

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.

Episerver

Introduction to 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

Integrating Data – 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.

Integrating Data – Dynamic Data Store

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

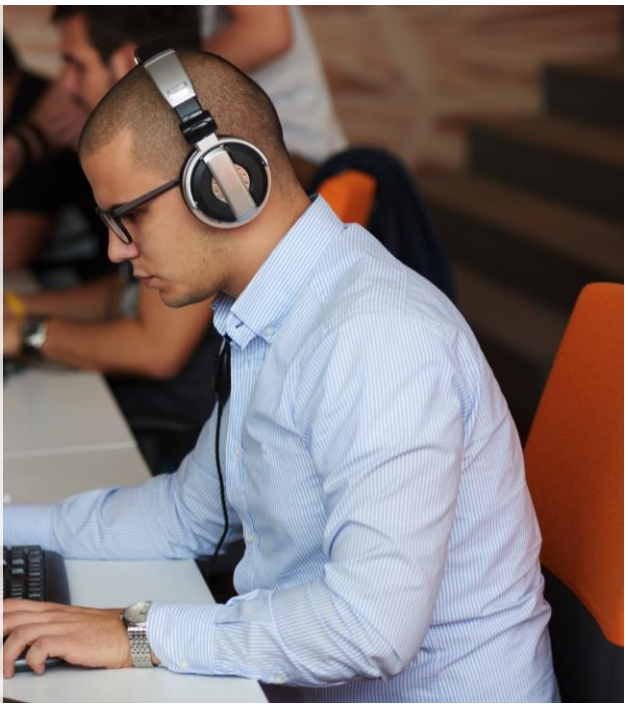
The columns whose name starts with "Indexed" have database indexes created on them.

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.

Integrating Data

Exercises for Module D

1. Implementing favorite pages with DDS
2. Implementing a Partial Route



Episerver

Episerver CMS Advanced Development

Rendering and Personalizing 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.
Understand how content renderers are selected => Learn how to control the rendering of content

Episerver

Rendering and Personalizing Content

Topics

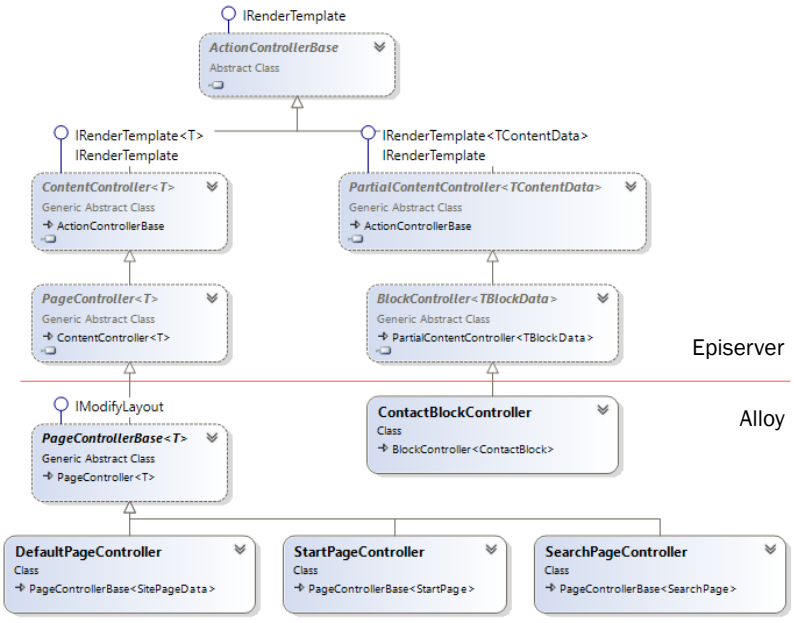
- Rendering templates
- Display channels
- Resolving templates
- Personalizing with visitor groups

Rendering and Personalizing Content – Rendering templates

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.



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.

Rendering and Personalizing Content – Rendering templates

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.

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” })]

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.

Most sites use both.

Rendering and Personalizing Content – Display channels

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

Rendering and Personalizing Content – Display channels

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

Episerver

Rendering and Personalizing Content – Display channels

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

```
[TemplateDescriptor(Default = true)]
public partial class MyBlockControl : BlockControlBase<MyBlock>
{ }

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

Episerver

Rendering and Personalizing 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();
```


Rendering and Personalizing Content – 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")
    });
```

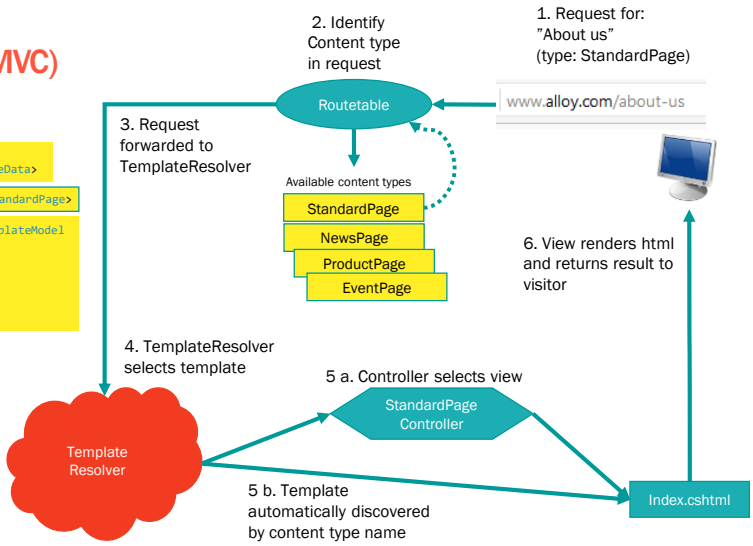
Views

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

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

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

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



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.

Rendering and Personalizing Content – Resolving templates

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).

Rendering and Personalizing Content – Resolving templates

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”

Episerver

Rendering and Personalizing Content – Resolving templates

Single template, multiple renderings

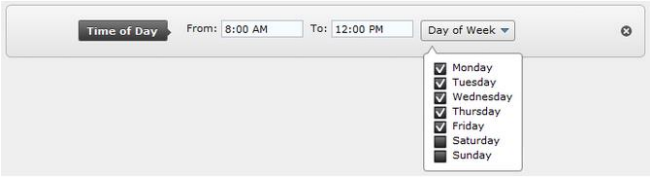
- The following example shows how to set CSS class depending on the active channel

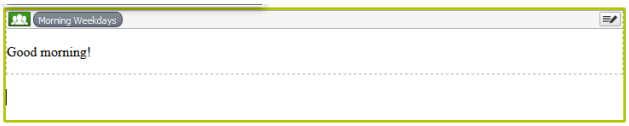
```
bool mobileDisplayChannelActive =
        ServiceLocator.Current.GetInstance<DisplayChannelService>()
            .GetActiveChannels(this.ControllerContext.HttpContext)
            .Any(c =>String.Equals(c.ChannelName,"mobile",StringComparison.OrdinalIgnoreCase));

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

Episerver

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.



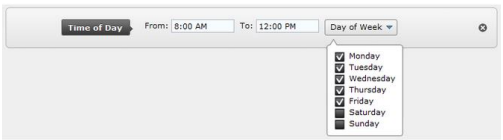


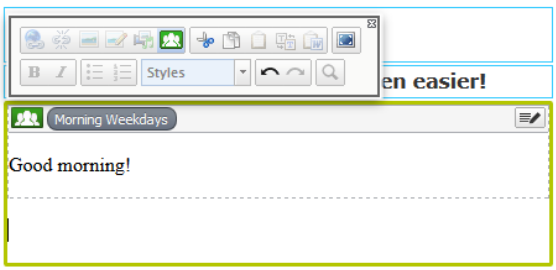
Rendering and Personalizing Content – Personalizing with visitor groups

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

Rendering and Personalizing Content – Personalizing with visitor groups

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

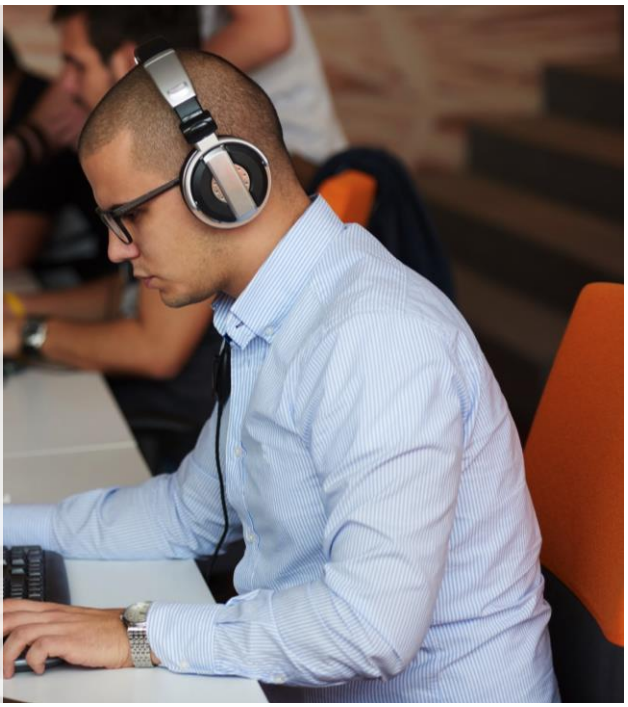
- Model class + Criterion class

```
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
    }
}
```

Rendering and Personalizing Content

Exercises for Module E

1. Using UIHints to apply display templates
2. Creating a PDF display channel
3. Detecting visitor groups with cookies



Episerver

Episerver CMS Advanced Development

Localizing for Editors and Visitors

EPiServer makes it easy to set up and work with several languages both in the UI and for the visitor. It is important to know how to set up a language handling structure that is suitable for your specific site. Localization is the process of customizing your application for a given culture and locale.

Episerver

Localizing for Editors and Visitors

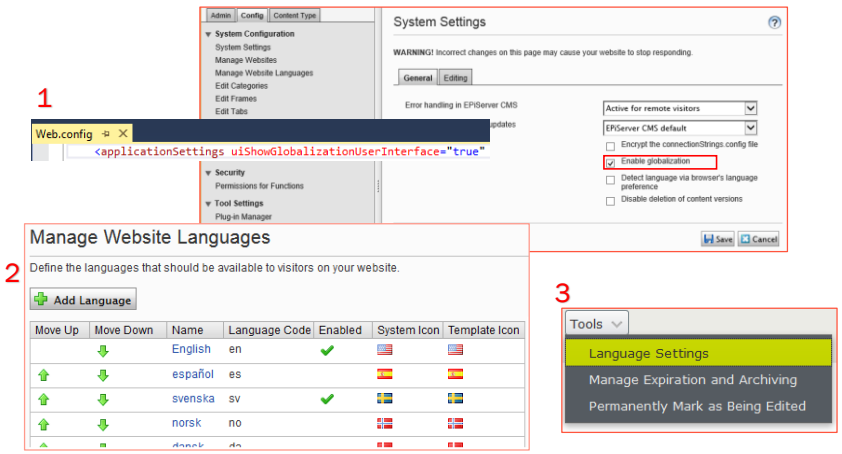
Even if you don't intend to localize into more than one language, you should still learn how to use localization resource files because they override code and administrator settings.

Topics

- Translate the UI and content of a site
  - The difference between globalization and localization
- Set up a site with multiple languages … and get it right the first time
  - What is UI and what is content
  - Language resources
  - Master language
- Using the Episerver Languages Add-on
  - How to work with globalized content

Localizing for Editors and Visitors

Setting it up



Episerver

Setting it up

Globalization on the website is enabled in the <episerver> section in the web.config. The setting is also visible under “System Settings” in Admin (Config tab, System Configuration section). Changing the system settings (including Globalization) from Admin is not recommended since it will (try to) update the web.config file, which can cause problems, especially in a production environment.

Add the languages to be available for the website with “Manage Website Languages” in Admin.

Choose Add Language to add a new language

Select a language and then select the Enabled checkbox to enable a language on the website

Optionally: Set access levels if needed

Make the languages available for web editors with the Language Settings in Edit view.

Fallback and replacement languages can be used to display other languages for visitors on the website.

Localization system

- Localization service
  - Episerver.Framework.Localization.LocalizationService
- Provider-based
- Why
  - Remove the requirement to put XML files in a folder named lang under the web root.
  - Still support simple XML files in a web folder.
  - Make it easier to create testable code that uses localization.
  - Make the service replaceable and extendable.
- System localizations are embedded in the Episerver assemblies, but can be overridden

Translate using LocalizationService

- A look at Alloy; where are the xml files are located and where in the configuration file can you find the path (hint: <localization>).

- Retrieve the localization service:

```
using EPiServer.Framework.Localization;
var localizationService = ServiceLocator.Current.GetInstance<LocalizationService>();
```

- Then retrieve a localized string using GetString():

```
var searchButtonText = localizationService.GetString("/button/search");
```


Localizing for Editors and Visitors

Localization Provider Configuration

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

Html.Translate()

- Used to translate language-specific strings
- Supply a simplified XPath expression to indicate which string you want to retrieve from the Episerver language resource files
- Uses the LocalizationService behind the scenes
  - Example:

```
@Html.Translate("/button/search")
```

Never hard-code texts!

Localizing for Editors and Visitors

Checking page languages from code

- Get all language branches

```
PageDataCollection allLanguages = DataFactory.Instance.GetLanguageBranches(myPageReference);
```

- Check if page exists in specific language

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

- In Episerver version 7+, use ContentRepository, not DataFactory:

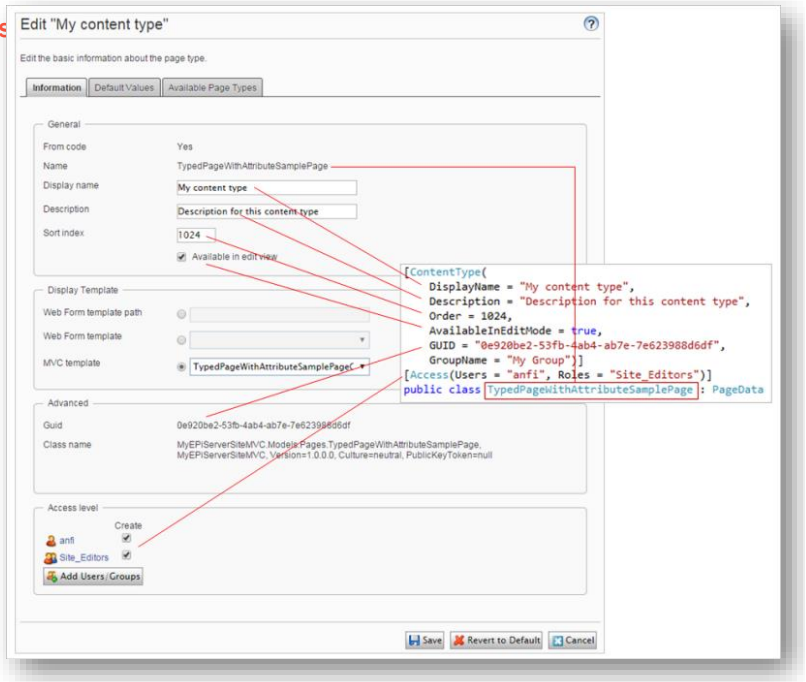
```
var contentRepository =
ServiceLocator.Current.GetInstance<IContentRepository>();

contentRepository.GetLanguageBranches<PageData>(PageReference.StartPage);
```

Content types in code/admin

Initial content type metadata should be set in code using [ContentType] and [Access] attributes. Values must be literal or constant expressions.

Admins can override using UI, which updates the metadata in the database.

Admins can revert back to the initial (code) values by clicking **Revert to Default** button.



Localization of content types

To use automatic localization of the content type metadata create one or more XML files in the **~/Resources/LanguageFiles** folder (with any name).

You could create one file for everything in all languages, or one file per feature and per language, or any combination.

Create the XML as follows:
http://world.episerver.com/blogs/Linus-Ekstrom/Dates/2013/12/New-standardized-format-for-content-type-localizations/

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
    <language name="English" id="en">
        <contenttypes>

            ...

    <language name="Dansk" id="da">
        <contenttypes>
```

Localization of custom content types and properties

Underscored element names are custom content type names and property names defined by your site:

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
    <language name="English" id="en">
        <contenttypes>
            <newspage>
                <name>News</name>
                <description>A news page describes recent events.</description>
                <properties>
                    <eventlisting>
                        <caption>Event Listing</caption>
                        <help>A list of events.</help>
                    </eventlisting>
```


Localizing for Editors and Visitors

Localization of content type default values

You can set default values for interfaces and base classes that will then be used by all custom types that implement or inherit from them:

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
    <language name="English" id="en">
        <contenttypes>
            <icontentdata>
                <properties>
                    <disableindexing>
                        <caption>Disable indexing</caption>
                        <help>Prevents the content from being indexed by Google.</help>
```

Localization of group/tab names

You can localize group/tab names like this:

```
<?xml version="1.0" encoding="utf-8"?>
<languages>
    <language name="English" id="en">
        <groups>
            <sitesettings>Site Settings</sitesettings>
            <eventinfo>Event Info</eventinfo>
```


Localization of Episerver user interface

You can localize other parts of the Episerver user interface like this:

http://www.kimgunnarsson.se/working-with-categories-in-episerver-cms-8-a-quick-heads-up/


```
<?xml version="1.0" encoding="utf-8"?>
<languages>
    <language name="English" id="en">
        <admin>
            <categories>
                <tabledescription>Description</tabledescription>
```

Localizing for Editors and Visitors

Content areas and blocks on a globalized site

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

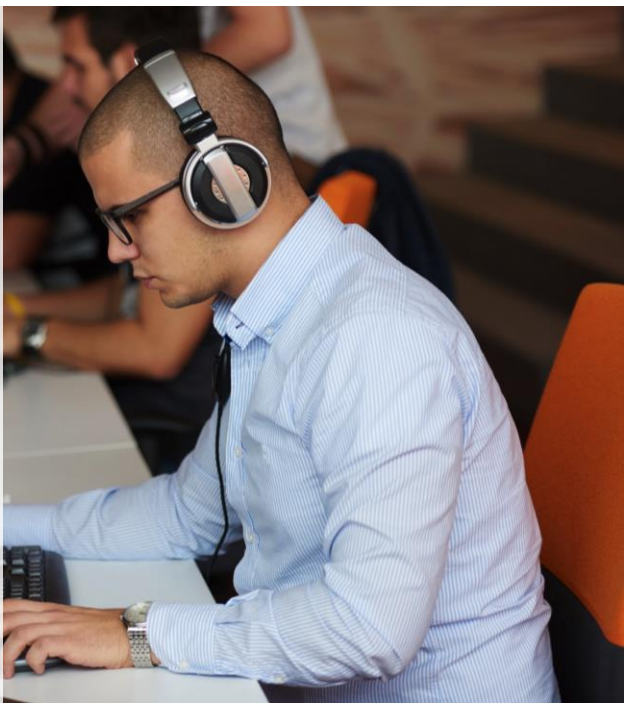
- What does the `[CultureSpecific]` attribute do when applied to:
  - A property of type **string** or **XhtmlString**?
  - A property of type **ContentArea**?

Episerver

Localizing for Editors and Visitors

Exercises for Module F

1. Localizing the TinyMCE toolbar
2. Localizing content types for editors
3. Localizing selection factories
4. Localizing content for visitors
5. Automating translations with Episerver Languages



Episerver

Episerver CMS Advanced Development

Indexing Content using Search and 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.

Indexing Content using Search and Find

Topics

- Episerver Search (integrated in CMS)
- Episerver Find

Episerver Search

- Episerver Search is a very simple but effective search solution and it should cover the needs of any basic search.
- The base is built upon the Lucene indexer.

Indexing Content using Search and Find – Episerver Search

Searching indexed content with CMS Search (Lucene)

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

```
private readonly SearchHandler searcher;

var query = new EPiServer.Search.Queries.Lucene.FieldQuery("alloy");
SearchResults results =
    searcher.GetSearchResults(query, page: 1, pageSize: 10);
```

Built-in features

- Full-text search
- Global search
- Static facets
- Event driven indexing
- Index any type of content
- Access-based search result filtering
- Pluggable search interface
- Instant search

Indexing Content using Search and Find – Episerver Search

Limitations and load balancing

- Search will not index blocks in content areas (by default).
- In a load balanced environment, the supported scenario is to install the search service on one of the machines, and configure that machine in the search settings for other machines.
- If more advanced options are needed use Episerver Find

Indexing Content using Search and Find – Episerver Search

Installing and deploying the Search Service

- The Search service is deployed through Nuget package Episerver.Search.
- Configure client for search by setting baseUri in <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>
```

Episerver

Indexing Content using Search and Find – Episerver Find

Why use Episerver Find?

- Increase organic traffic with automatic/custom landing pages
  - An “Automatic” or “Custom” landing page is content (for example a page type instance if you talk Episerver CMS) which renders differently depending on user input. A kind of “personalized content-type”. A google hit can for example point to an automatic landing page, which will show content related to the google search.
- Amplify visitor engagement with adaptive navigation
- Boost conversions with guided search

Episerver

Episerver Find features

- Multi-language stemming
- Best bets
- Related queries
- Highlighted summaries
- Autocomplete
- Search as you type
- Search in files/attachments
- Custom weighting of results
- Statistics and search optimization
- Find Connectors
- Deconstruction of words (Swedish and Norwegian)

Indexing Content using Search and Find – Episerver Find

Product options

- Software-as-a-service
- Virtual Appliance
  - The Virtual Appliance is a virtual server that the customer host in their own network infrastructure. There are two versions; one that has a VPN tunnel to Episerver Hosting and one that doesn't.

Installing

- Episerver CMS 6 and higher
- Support for Episerver Commerce
- Requires the full .NET framework (not Client Profile)
- Depends on JSON.NET (Newtonsoft.Json.dll)
- Installed through NuGet
- Requires additional license + index

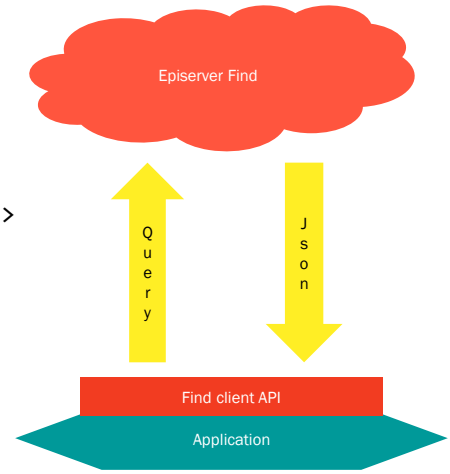
Indexing Content using Search and Find – Episerver Find

Getting started

1. Create an account
2. Create a Developer Service
3. Download.NET API or Episerver integration
4. Add your index information to app.config/web.config
5. Index and search for object (strongly typed)

Using Find from generic application

```
var result = client.Search<T>
    .For("some search term")
    .GetResult();
```



Indexing Content using Search and Find – Episerver Find

Episerver Find Configuration

```
<configSections>
  <section name="episerver.find"
   type="Episerver.Find.Configuration,
         Episerver.Find" />
</configSections>

<episerver.find
  serviceUrl="http://... "
  defaultIndex="myindex"/>
```

IClient

- Handles all interactions with the search engine
  - Indexing
  - Deleting
  - Updating
  - Get by ID
  - Searching

Indexing

```
using Episerver.Find;

IClient client = Client.CreateFromConfig();
var someObject = new SomeClass();
var result = client.Index(someObject);
```

Indexing is done using the clients 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.

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.

Indexing Content using Search and Find – Episerver Find

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.

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.

Summary: CRUD

- The Index(…) method adds and updates documents
- The Get<T>(…) method retrieves objects by ID
- The Delete<T>(…) method deletes objects by ID

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.

Indexing Content using Search and Find – 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

Indexing Content using Search and Find – 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.

Episerver

Indexing Content using Search and Find – 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.

Episerver

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.

Indexing Content using Search and Find – Episerver Find

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>**.

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.

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

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.

Indexing Content using Search and Find – 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")
```

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.

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.

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

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. For more extensive examples see:
http://find.episerver.com/Documentation/dotnet-api-filtering

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().
For more information see:
http://find.episerver.com/Documentation/dotnet-api-custom-filtering-methods

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

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)

Indexing Content using Search and Find – Episerver Find

Terms facets

```
var authorCounts =
    result.TermsFacetFor(x => x.Author);

foreach(var author in authorCounts)
{
  Console.Write(author.Term
    + " (" + author.Count + ")");
}
.GetResult();
```

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.

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

Indexing Content using Search and Find – 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.

Indexing Content using Search and Find – 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.

Indexing Content using Search and Find – 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.

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.

Indexing Content using Search and Find – 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 UnifiedSearchHit objects. Further, it's also possible to add filters that should be applied specifically for documents corresponding to an added type.

Unified Search – searching

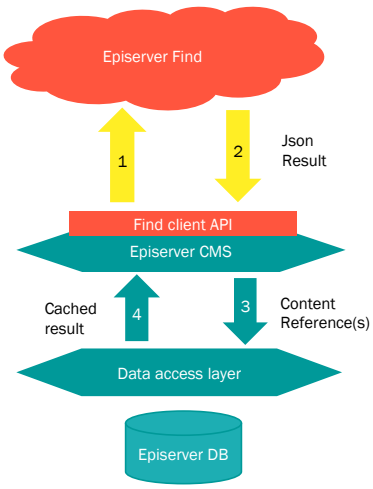
```
client.UnifiedSearch(Language.English)
  .GetResult();
   // is equivalent to
client.Search<ISearchContent>(
  Language.English)
  .GetResult();

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 result = SearchClient.Instance
    .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)
{

}
```

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.

Indexing Content using Search and Find – 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.

Indexing Content using Search and Find – Episerver Find

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

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**.

Indexing Content using Search and Find – 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.

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.

Indexing Content using Search and Find – Episerver Find

Expanding Search

- All features are OPT-IN
- Needs to be enabled through code
- For example:
  - Statistics via `Track`()
  - Best Bets via `ApplyBestBets`()
  - Synonyms via `UsingSynonyms`()

Indexing Content using Search and Find – Episerver Find

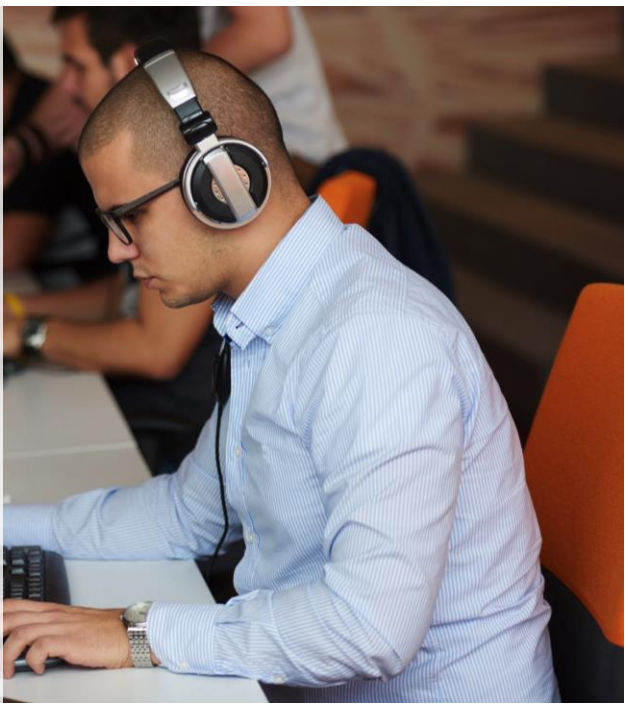
Learn more about Episerver Find

- Episerver Find for developers course (1-day course)
- find.episerver.com
- find.episerver.com/documentation
- world.episerver.com/forum
- flyfind.demo.episerver.com
- find.demo.episerver.com

Indexing Content using Search and Find

Exercises for Module G

1. Adding Fields to Episerver Search
2. Configuring Episerver Find
3. Implementing Episerver Find for CMS



Episerver CMS Advanced Development

Extending with Add-ons

With Episerver (from version 7), extensions to your site can be installed via the Add-on store and 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

Extending with Add-ons

Topics

- The modules Shell and CMS
- What are add-ons
- Creating an add-on
- Examples of Episerver add-ons

Episerver

Why extend Episerver CMS?

For visitors to the website:

- Develop templates and user controls 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.

Extending with Add-ons

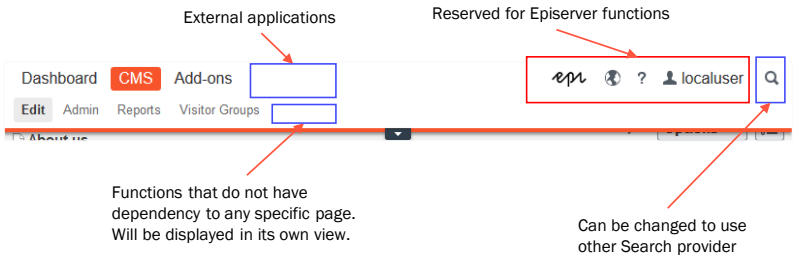
Examples of extension points

- Edit View: Components
- Admin: Plug-ins
- Visitor group criteria
- Dynamic Content
- Scheduled jobs
- Custom property types

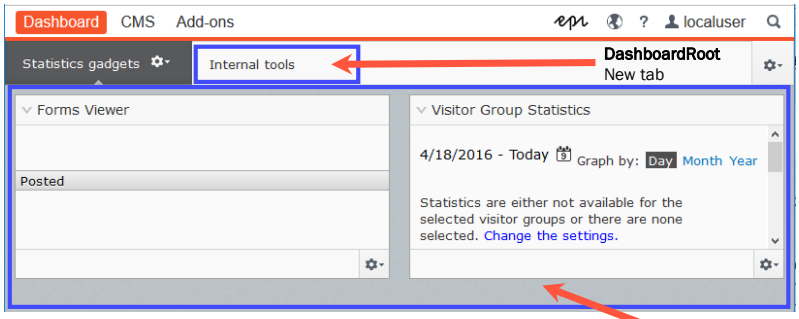
Extending using Episerver plug-ins

- Episerver plug-ins offer a flexible way to extend Episerver using .NET attributes
- Developing plug-in's
- 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.

Extending the Global menu

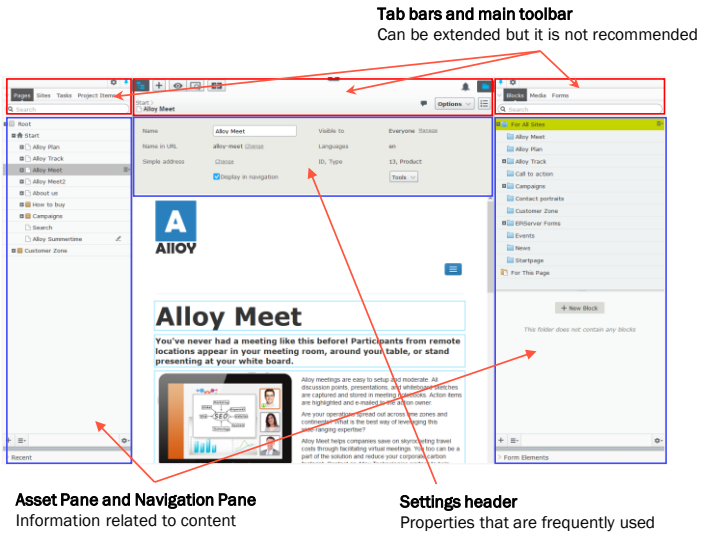


Extending the Dashboard



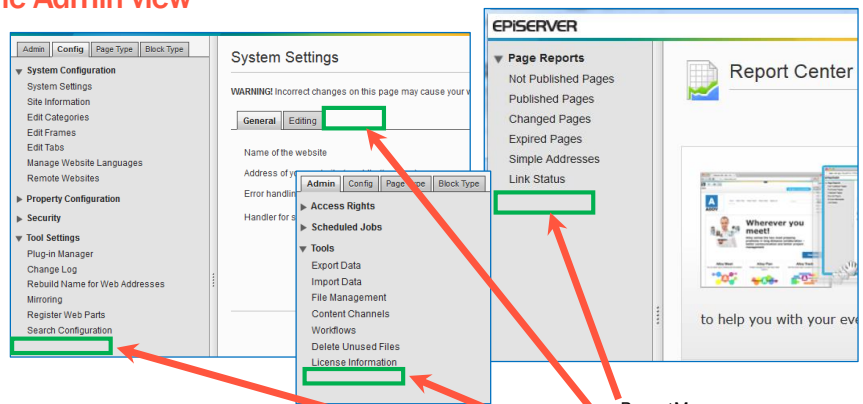

```
[Component(Title="My Dashboard plug-in",
           Description="A plug-in",
           PlugInAreas=PlugInArea.DashboardDefaultTab,
           Categories="dashboard")]
```


Extending the Edit view



Extending with Add-ons

Extending the Admin view



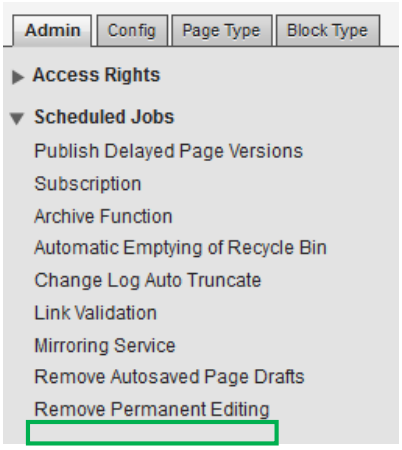

```
[GuiPlugIn(DisplayName="My Plug-in",
           Description="A plug-in",
           Area=PlugInArea.AdminConfigMenu,
           Url="~/PlugIns/MyPlugIn")]
```

ReportMenu
SystemSettings
AdminMenu
AdminConfigMenu

Episerver

Extending with Add-ons

Extending with Scheduled Jobs




```
[ScheduledPlugIn(DisplayName = "MyScheduledJob")]
public class MyScheduledJob
{
    public MyScheduledJob()
    {
    }

    /// <summary>
    /// Starts the job
    /// </summary>
    /// <returns>A status message that will be logged</returns>
    public static string Execute()
    {
        //Add implementation
        return "OK";
    }
}
```

Episerver

Scheduled jobs (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;
```


Extending with Add-ons

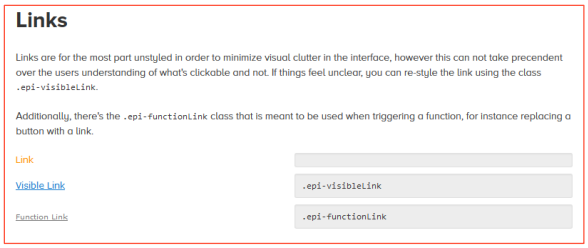
Scheduled jobs (2 of 2)

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!";
}
```

Extending with Add-ons

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.

Episerver

Extending with Add-ons

What is the Shell? What are modules?

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

Episerver CMS is shipped with three shell modules.

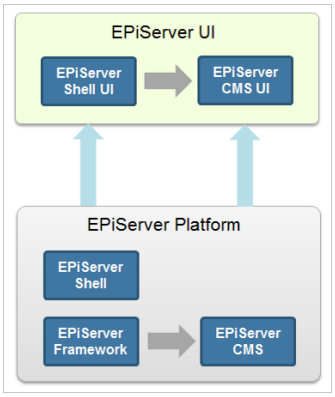
Shell: dashboard and resources shared between multiple products

CMS: shell resources specific to the CMS

Episerver.Packaging.UI: interface files for the Add-on store

Public and protected modules

Different root path, protected modules is configured by a location block to only allow authenticated editors



Episerver

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.

Extending with Add-ons

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

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.

Extending with Add-ons

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.

Episerver

Extending with Add-ons

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.

Episerver

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. (*)

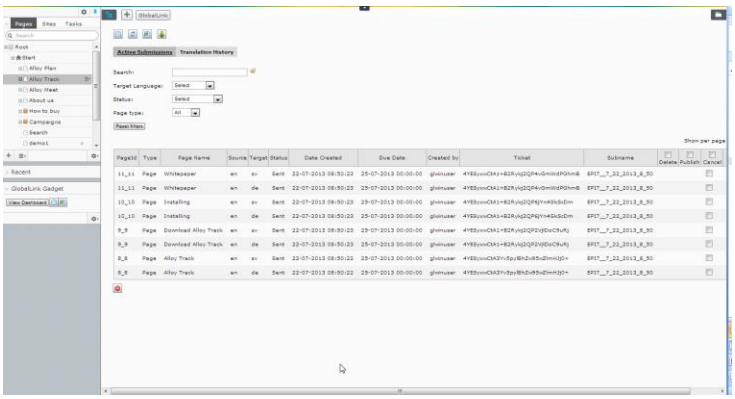
Extending with Add-ons

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/



Extending with Add-ons

Episerver Social Reach

- With Social Reach you can set up your different social channels and set up 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.

Extending with 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.

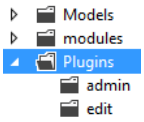
Extending with Add-ons

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/



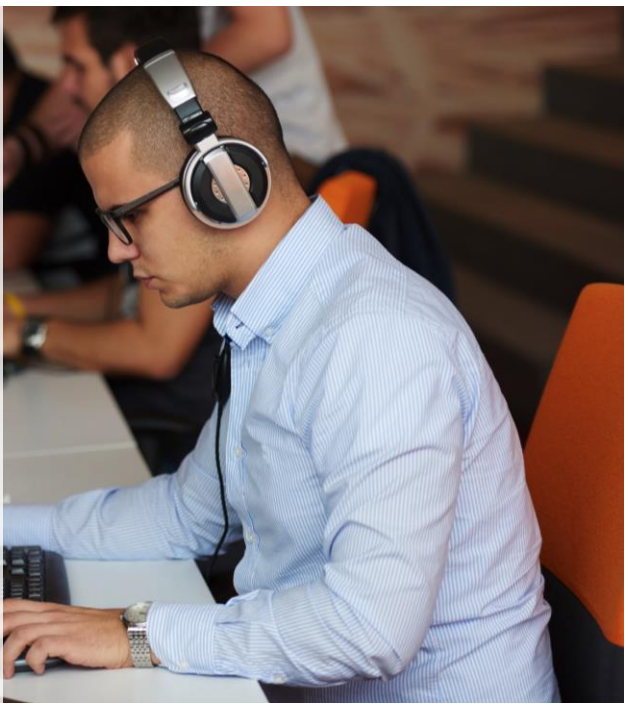

```
<!-- Restrict access to files beneath the Plugins folder -->
<location path="Plugins/admin">
  <system.web>
    <authorization>
      <allow roles="WebAdmins, Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
<location path="Plugins/edit">
  <system.web>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

Episerver

Extending with Add-ons

Exercises for Module H

1. Exploring existing add-ons and plug-ins
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

Episerver CMS Advanced Development

Conclusion

Episerver