

Bruce Gordon

Episerver

5/23/2019

Episerver Workshop for .NET Developers



Contents

[Setting up the developer environment 1](#_Toc9338979)

[Installing Episerver Extension 1](#_Toc9338980)

[Configuring the Episerver NuGet Package Source 1](#_Toc9338981)

[Creating a new empty site project 1](#_Toc9338982)

[Initialization module to add Admin account 1](#_Toc9338983)

[Defining Content Types 2](#_Toc9338984)

[Defining a Start page type 2](#_Toc9338985)

[Creating a Start page instance 3](#_Toc9338986)

[Creating a Start page template 7](#_Toc9338987)

[Enabling upload of any file by defining a media type 9](#_Toc9338988)

[Uploading files 10](#_Toc9338989)

[Customize handling of images by defining an image type 12](#_Toc9338990)

[Uploading images 13](#_Toc9338991)

[Inheriting Content Types 16](#_Toc9338992)

[IList<string> Custom Display Template (Optional) 20](#_Toc9338993)

[Defining Blocks 21](#_Toc9338994)

[Creating an editorial block type 21](#_Toc9338995)

[Dependency Injection 23](#_Toc9338996)

[Out-of-the-box StructureMap 23](#_Toc9338997)

[Setting up MVC Controller Dependency Injection 24](#_Toc9338998)

[Personalization 27](#_Toc9338999)

[Setting up a Visitor Group 27](#_Toc9339000)

[Adding Geographic Visitor Group Criteria 31](#_Toc9339001)

[Getting Geographic Criteria to work with Localhost 33](#_Toc9339002)

Copyright © 1996-2018 Episerver AB. All rights reserved.

Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without expressed written permission of Episerver. Episerver assumes no liability or responsibility for any errors or omissions in the content of this document.

Episerver is a registered trademark of Episerver AB, Episerver Inc. and other global subsidiaries.

# Setting up the developer environment

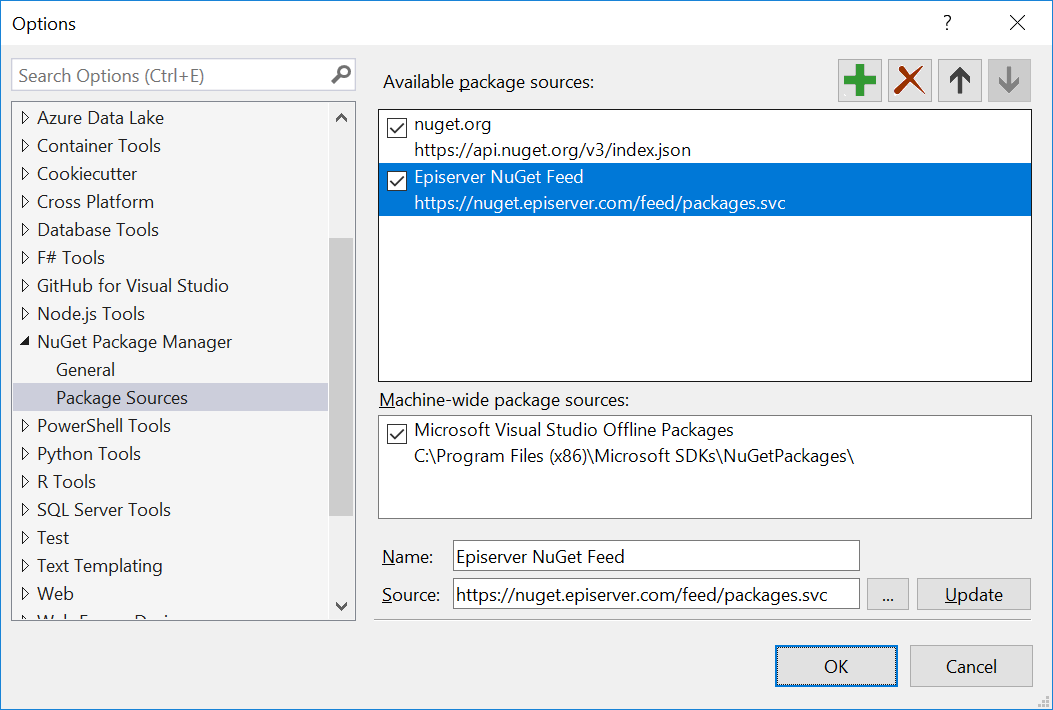
You can find the solution code for all the demos in this workbook at, <https://github.com/bruceEgordon/Epidemosite>. At the root of the solution folder you’ll also find a folder named “**Assets**” that contains some random media and image files. You’ll use those files in the site you build by following this guide, so make sure you download them.

Setting up a development environment for Episerver CMS doesn’t require very much effort. As long as you have a version of Visual Studio with the ASP.NET and SQL Express components installed, the only thing you’ll need to add is an Episerver plugin for Visual Studio.

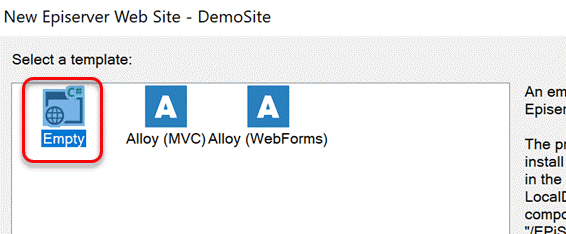
## Installing Episerver Extension

1. Start Microsoft Visual Studio.
2. In Visual Studio 2015 or 2017, navigate to **Tools** | **Extensions and Updates…**, or in Visual Studio 2019, navigate to **Extensions | Manage Extensions**.
3. In the section on the left, click **Online**, to show the Visual Studio Marketplace.
4. Press **Ctrl + E**, or click in the Search box, and then enter **Episerver**.
5. Select the **Episerver CMS Visual Studio Extension**, click **Download**.
6. Wait for the extension to download, and then click **Close**.
7. Close Visual Studio and wait for the VSIX Installer to start.
8. Click **Modify**, wait for it to complete installing, and then click **Close.**

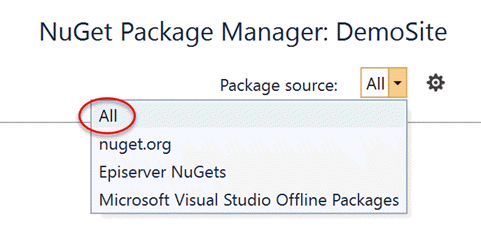
## Configuring the Episerver NuGet Package Source

1. Start Microsoft Visual Studio.
2. Navigate to **Tools** | **NuGet Package Manager** | **Package Manager Settings**.
3. In the **Options** dialog, in the list on the left, click **Package Sources**.
4. If the EpiserverNuGet feed does not exist as an available package source, as shown in the following screenshot, then click the **green plus** button to add it. The name can be anything, although we recommend using **Episerver NuGets**, and the path must be: https://nuget.episerver.com/feed/packages.svc/

## Creating a new empty site project

1. Launch Visual Studio and create a new project using the **Episerver Web Site** template, name it **DemoSite**.
2. In the **New Episerver Web Site** dialog choose the **Empty** template and click **OK**:  
   
3. Right-click the **DemoSite** in Solution Explorer and choose **Manage NuGet Packages…**

**Remarks:** Note the Episerver packages the makeup the core CMS framework. Technically you don’t have to use the Episerver project template, you could take a blank MVC site and manually add the NuGet packages, add the necessary Web.config entries as well as modify the global startup. Using the template does all of this for you.

1. Change the NuGet **Package source** drop-down menu to **All**:  
   
2. From the **Browse** tab find and install **EPiServer.Search**, **EPiServer.Search.Cms** and **Bootstrap**.

**Remarks:** This step is mostly to show how to add packages but this will also allow the use of Global search and search boxes in the **Navigation** and **Assets** panes in the CMS edit view.

1. Click the **Updates** tab in the Package Manager window and change the **Package source** from **All** to **Episerver NuGets**.

**Remarks:** Episerver implements a continues release cycle so it is very likely that even if you just installed the latest Episerver plugin for Visual Studio the project template is still referencing older packages and you will find newer version available. Note that you could update everything not just the Episerver packages, this demo is just focusing on the Episerver ones.

1. If there are any Episerver packages available to be updated go ahead and update all the EPiServer packages.

**Remarks:** If you run the site immediately after applying updates it’s possible you’ll receive an error about the database needing to be updated. If any of the Episerver packages require database schema changes than you’ll need to update the database, that’s the next step.

1. From the Visual Studio menu choose **Tools | NuGet Package Manager | Package Manager Console**.
2. Run **Update-EPiDatabase** from the Package Manager console.

## Initialization module to add Admin account

The Episerver empty site template is setup to use Forms based authentication using either users defined in the Episerver database was well as Windows accounts. The following steps will create an admin account in the Episerver database to avoid using your Windows accounts. Episerver also supports logins from external authentication with OWIN based authentication. The Alloy site template is configured with OWIN but still uses the AspNetIdentity where users are defined in the Episerver database.

1. Right-click the **Business** folder and choose **Add -> New Folder** and name it **Initialization**.
2. Right-click the new **Initialization** folder and choose **Add -> New Item**
3. In the **Add New Item** dialog choose **Episerver** from the tree and then choose **Initialization Module** from the list of choices, name it **AdminAccountInitializationModule.cs**.
4. Add the following code snippet into the **Initialize** method and fix any missing namespaces:

string storedRoleName = "WebAdmins";

string userName = "Admin";

string password = "Pa$$w0rd";

string email = "admin@alloy.com";

MembershipUser user = Membership.GetUser(userName);

if(user == null)

{

Membership.CreateUser(userName, password, email);

if (!Roles.RoleExists(storedRoleName))

{

Roles.CreateRole(storedRoleName);

}

Roles.AddUserToRole(userName, storedRoleName);

}

**Remarks:** This code is just for getting a SQL account that you can initially log in with. Optionally, you can use a local Windows account if it is a member of the Administrators Windows group. The **WebAdmins** role/group has default permissions along with the Windows Administrator group. Adding this new “Admin” account ensures you can manage the site with it. Once this code has been run, you can remark it out, but since it only creates the “Admin” role if it doesn’t exist, you can safely leave it if you like. Obviously, this is not what you would want to leave in production code.

1. Start the site but be prepared to receive an initial **404** response since you don’t yet have any content.
2. Manually navigate by typing **/EPiServer/CMS/** to the end of the URL and confirm that you can log in as a CMS admin using the password defined in the code above.
3. Close the browser.

**Remarks:** At this point, you don’t have any user content to browse to. The next step is to create a content type that will be used to define a basic home page for the site.

# Defining Content Types

## Defining a Start page type

1. In **DemoSite**, expand **Models**, right-click **Pages**, and click **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog window, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **StartPage.cs** for the name, and click **Add**.
3. In the [ContentType] attribute, change the **DisplayName** to **Start**.
4. In the [ContentType] attribute, add a **GroupName** attribute and set it to **“Specialized”** and add the sort index (i.e. **Order**) within that group to **10**.
5. In the [ContentType] attribute, set the **Description** of “The home page for a website with an area for blocks and partial pages.”
6. In the body of the class, uncomment the **MainBody** property with all its attributes and change its **Order** to **20**.
7. Add a string **Heading** property with appropriate attribute values. **Order** values are used to sort properties within a tab group. It is good practice to use multiples of ten so that future properties can be added between existing ones.
8. Add a content area property with appropriate attribute values.

Your complete page type class should look something like the following code:

using EPiServer.Core;

using EPiServer.DataAbstraction;

using EPiServer.DataAnnotations;

using System.ComponentModel.DataAnnotations;

namespace DemoSite.Models.Pages

{

[ContentType(DisplayName = "Start",

// your code will have a random GUID here

GroupName = "Specialized", Order = 10,

Description = "The home page for a website with an area for blocks and partial pages.")]

public class StartPage : PageData

{

[CultureSpecific]

[Display(Name = "Heading", Description =

"If the Heading is not set, the page falls back to showing the Name.",

GroupName = SystemTabNames.Content, Order = 10)]

public virtual string Heading { get; set; }

[CultureSpecific]

[Display(Name = "Main body",

Description = "The main body uses the XHTML-editor so you can insert, for example text, images, and tables.",

GroupName = SystemTabNames.Content, Order = 20)]

public virtual XhtmlString MainBody { get; set; }

[Display(Name = "Main content area",

Description = "The main content area contains an ordered collection to content references, for example blocks, media assets, and pages.",

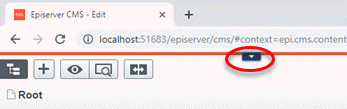
GroupName = SystemTabNames.Content, Order = 30)]

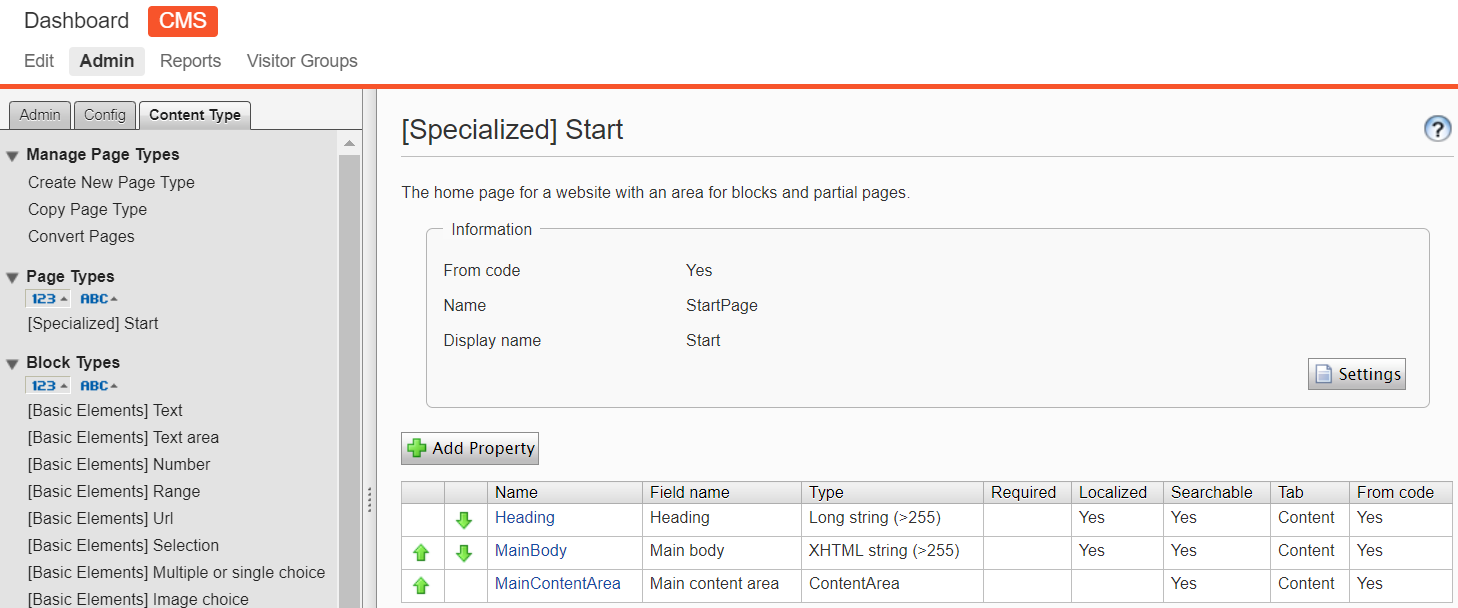
public virtual ContentArea MainContentArea { get; set; }

}

}

## Creating a Start page instance

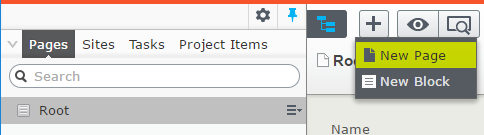
1. Start the website by navigating to **Debug** | **Start Without Debugging** or press *Ctrl* + *F5*.
2. Manually type **/EPiServer/CMS/** at the end of the URL address.
3. Log in asa CMS admin.
4. Click the arrow at the top of the page, in the middle, to open the top menu. Note that you can click it again to pin it, so it stays visible:  
   
5. Navigate to **CMS** | **Admin** | **Content Type**, click **Start**, and note that the synchronization process has registered your page type, including its properties, as shown in the following screenshot:

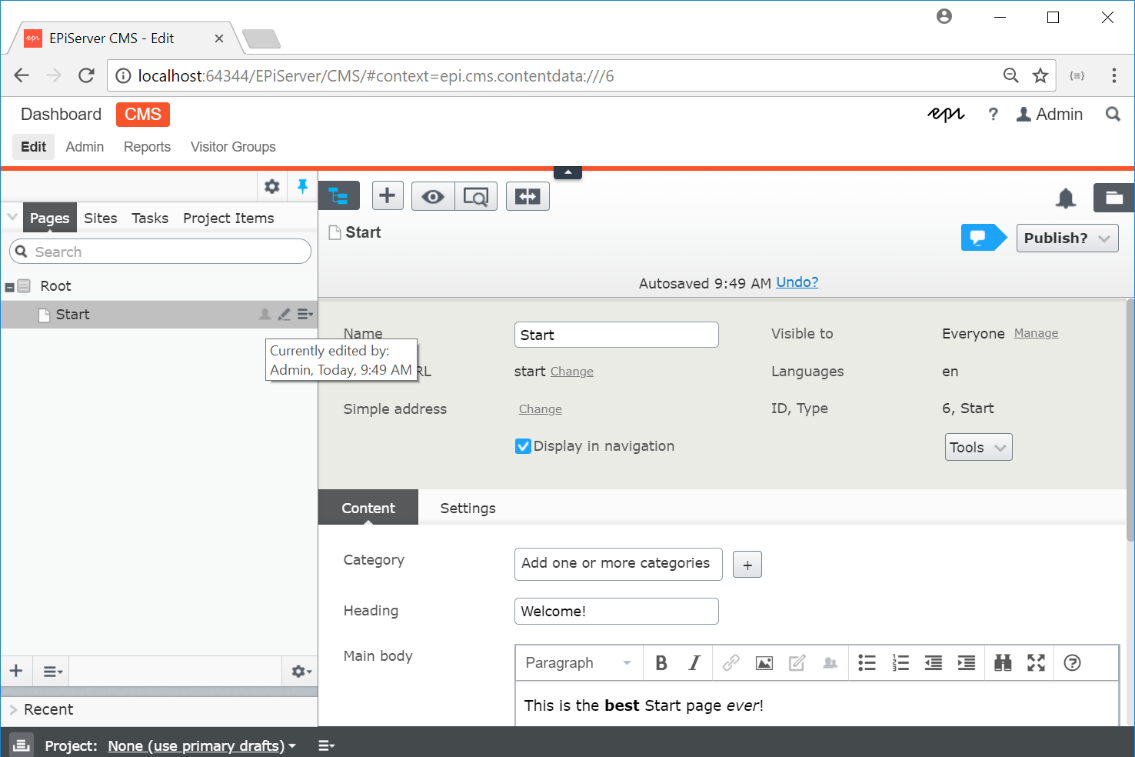


* Members of the virtual role **CmsAdmins** can modify many of the values that are initially set by your code attributes by clicking **Settings**. These changes will override your code, even if you subsequently change your code. **CmsAdmins** can use the **Revert to Default** button to reset to the code attribute values.

**Remarks:** The **Admin** account still lacks permissions to modify the **Root** of the site but since the account is a member of **WebAdmins** you have permissions through **Web.config** location settings to access the **Set Access Rights** view and manage the **Root** permissions.

1. Navigate to **CMS** | **Admin** | **Admin**, click **Set Access Rights** and use the **Add Users/Groups** to add the **WebAdmin** group to the list and assign them the full list of permissions.  
   A screenshot of a cell phone

   Description automatically generated
2. Navigate to **CMS** | **Edit**, expand the **Navigation** pane, and note it only contains a **Root** page.
3. Click **+**, and then click **New Page**, or click the context menu for **Root** and click **New Page**, as shown in the screenshot:
4. Enter the name **Start**, and then click **OK**.
5. Note the following:
   1. Pencil icon indicates a saved draft, so the page is not published yet, and it won’t be visible to visitors, and the person icon can be hovered over to show who is editing the page.
   2. Blue information icon warns that you must publish this page for visitors to see changes.
   3. The three custom properties, **Heading**, **MainBody**, and **MainContentArea**, are grouped and sorted under the **Content** tab in **All Properties** view, as shown in the following screenshot:

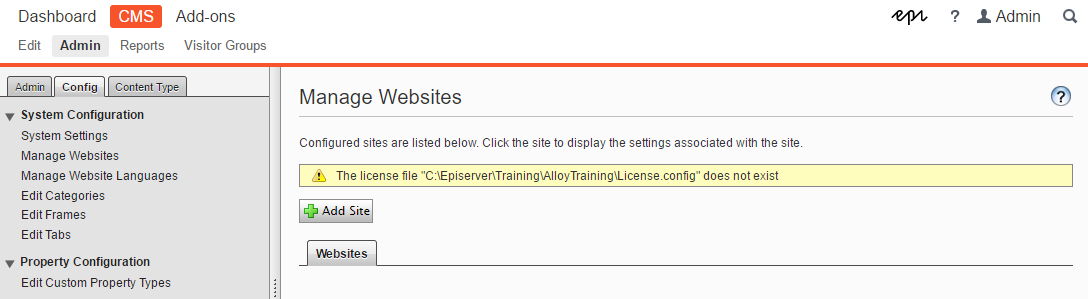


* There is no On-Page Editing (OPE) yet because you have not created a page template.

1. Enter values for **Heading** and **Main body**. Be creative!

* Every page has an inherited property named **Category**. If your editors do not want to use it, you can hide it. Our *Episerver CMS – Advanced Development* training course shows you how.

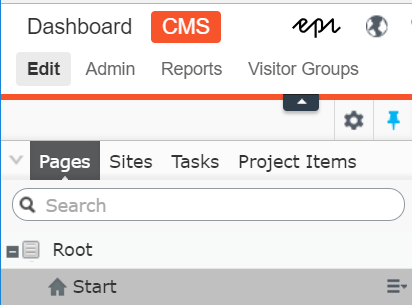
1. Click the **Publish?** button, and then click the green **Publish** button.
2. Navigate to **CMS** | **Admin** | **Config** | **Manage Websites**, and click **Add Site**, as shown in the following screenshot:



1. Enter the following details, as shown in the following screenshot:
   1. **Name**: Acme Site
   2. **URL**: [copy from browser address box]
   3. **Start page**: click […] to select the Start page you just created.
   4. **Use site-specific assets**: checked

A screenshot of a cell phone

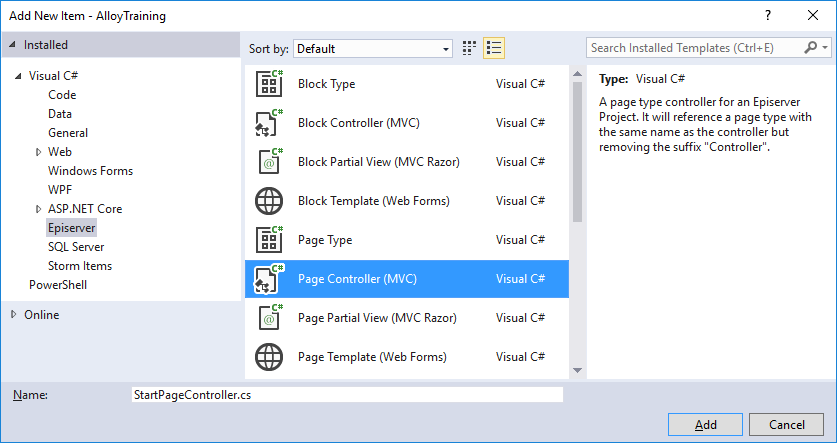
Description automatically generated

1. Click **Save**.
2. Switch to **Edit view** and note that the **Start** page now shows a “house” icon and the Global menu has a “globe” to switch to **Live view**, as shown in the screenshot.
3. Switch to **Live view** and note that the Start page still returns a 404 because it does not yet have a template.
4. Close the browser.

## Creating a Start page template

* A page template in MVC is a combination of a controller and a view.

1. In Solution Explorer, expand **Controllers**, and right-click and choose **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **StartPageController.cs** for the name, and click **Add**, as shown in the following screenshot:



1. Use the Quick Actions lightbulb to fix the missing namespace for the **StartPage** class. Make sure you pick the local project namespace, there’s a **StartPage** type in the **System.Web.WebPages** namespace that is **not** the one you want.

**Remarks:** Now you need a razor view for building out the display for the **Index** method of the controller. There is an Episerver template named **Page Partial View (MVC Razor)** you could use. I prefer to use the scaffolding feature of Visual Studio instead of the template provided by Episerver. It creates the folder for you as well as allowing you to select the model, but it will be missing the **div** holding a call to **@Html.PropertyFor(m => m.MainBody)** that is created by the Episerver template.

1. Right-click inside the **Index** method body and choose the **Add View…** menu option.
2. In the **Add View** dialog select **Empty** for the **Template** drop-down and select **StartPage** for the **Model class**, leave everything else at the default and click the **Add** button.

**Remarks:** Leaving the **Use a layout page** checked will result in Visual Studio generating a basic layout page for you. This is handy for demo purposes because it creates a basic navigation menu and links to Bootstrap and jQuery.

1. Below the **@model** declaration, replace the existing **<h2>** element code with the following:

<h1 @Html.EditAttributes(m => m.Heading)>

@(Model.Heading ?? Model.Name)

</h1>

<div>

@Html.PropertyFor(m => m.MainBody)

</div>

<div>

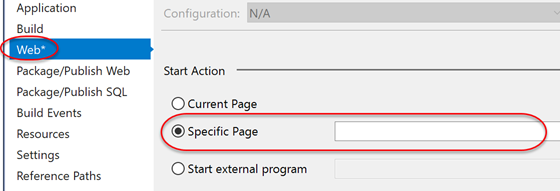
@Html.PropertyFor(m => m.MainContentArea)

</div>

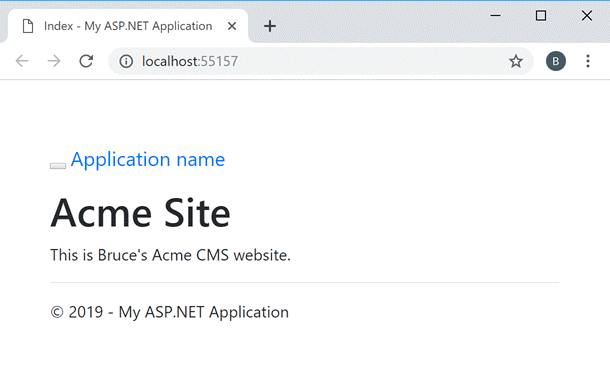
* The existing code uses **PropertyFor** to output the **MainBody** which enables the On-Page Editing (OPE) experience as well as correctly converting content GUID links to friendly URLs.

**Remarks:** If you run the site while editing the view you’ll probably get **404** error because Visual Studio tries to automatically browse to the view using a conventional path, CMS uses a different routing system. To avoid the error, you can set the **project properties** **-> Web -> Specific Page** to use a blank start page.

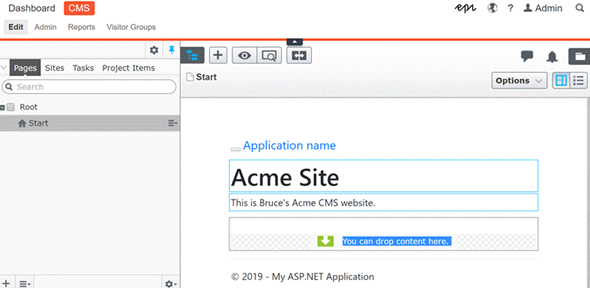
1. Right click the web site project in Solution Explorer and choose **Properties** from the menu.
2. Select the **Web** tab and then select **Specific Page** but leave the page field empty, it should look like the following image:



1. Save the Web settings and the start the site, it should look similar to the following image:



1. Enter **/EPiServer/CMS/** at the end of the address box, and log in as a CMS admin.
2. Navigate to **CMS** | **Edit**., and note the On-Page Editing (OPE) experience due to the page template and use of **PropertyFor** and **EditAttributes** in the view, as shown in the following screenshot:



1. Verify that the on-page edit logic for the <h1> tag works, i.e. that when you edit the text it is the **Heading** property value that is updated, and that when a value exists for **Heading** it is that value that is rendered to the visitor, otherwise the **Name** is rendered.
2. Empty the **Heading** property value and publish the change to verify that the fallback works, i.e. that the page name is displayed to the visitor and to the editor.
3. Close the browser.

## Enabling upload of any file by defining a media type

Before uploading asset files, a little code is needed for the system to be able to recognize media.

1. In Solution Explorer, expand **Models**, right-click **Media**, and click **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog window, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Media Type**, enter **AnyFile.cs** for the name, and click **Add**.
3. Change the **DisplayName** to **Any File**.
4. Add a **Description** of “Use this to upload any type of file.”
5. Delete the commented example property and the commented **MediaDescriptor** attribute.

Your code should look something like the following:

using EPiServer.Core;

using EPiServer.DataAbstraction;

using EPiServer.DataAnnotations;

namespace AlloyTraining.Models.Media

{

[ContentType(DisplayName = "Any File",

// your code will have a GUID

Description = "Use this to upload any type of file.")]

public class AnyFile : MediaData

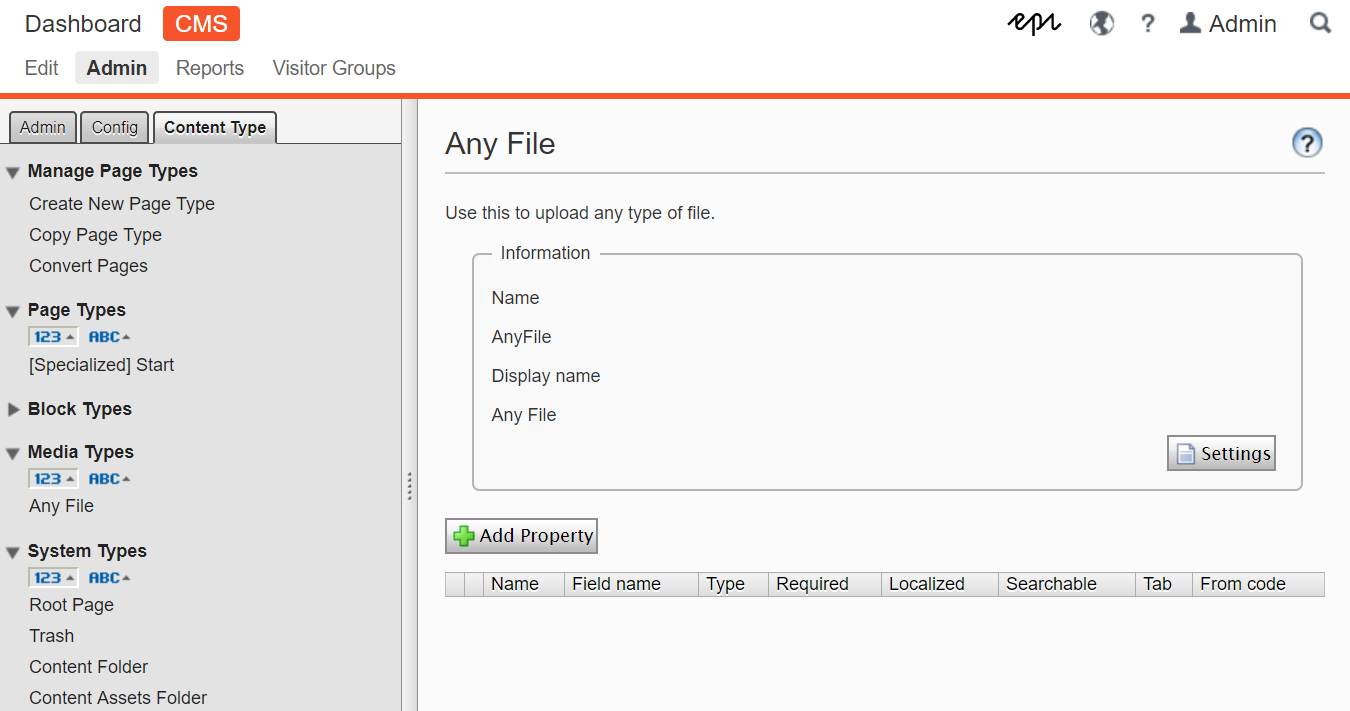
{

}

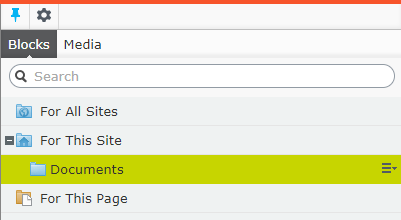
}

## Uploading files

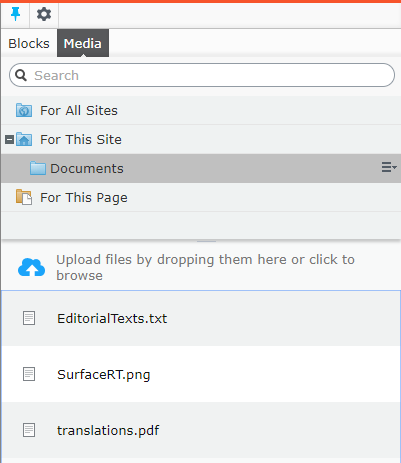
1. Start the website, navigate to **episerver/cms**, and log in as a CMS Admin.
2. Navigate to **CMS** | **Admin** | **Content Type**, and select **Any File**, as shown in the following screenshot:

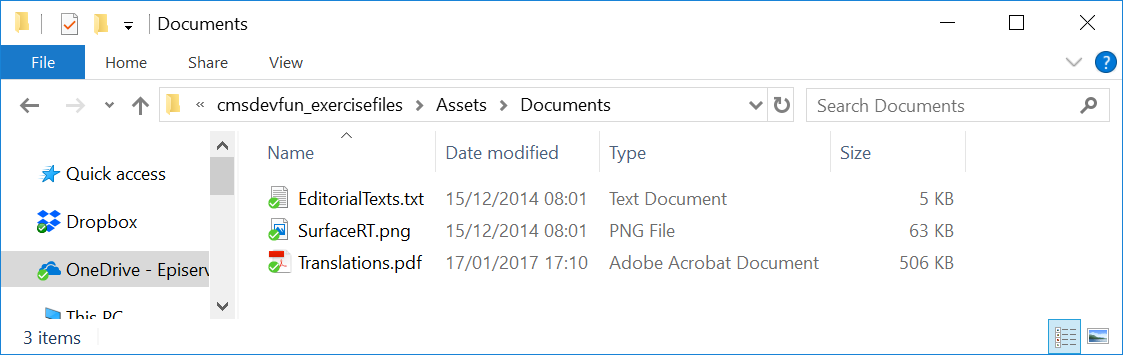


1. Navigate to **CMS** | **Edit**.
2. In **Assets** pane, create a folder named **Documents** in **For This Site**, as shown in the following screenshot:



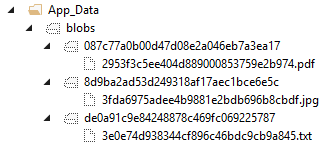
* If you cannot see the **For This Site** folder, then navigate to **CMS** | **Admin** | **Config** | **Manage Websites**, click the **Alloy Training** site, select the **Use site-specific assets** checkbox, and click **Save**.
* It does not matter if **Blocks** or **Media** is currently selected when you create a folder because they both share the same folder structure. But you must select the **Media** tab before you can upload files.

1. Select the **Media** tab.
2. Select the **Documents** folder.
3. Start **File Explorer**.
4. From the folder **<<source code project>>\Assets\Documents**, drag and drop **SurfaceRT.png**, **EditorialTexts.txt**, and **Translations.pdf** into the **Documents** folder, as shown in the following screenshots:

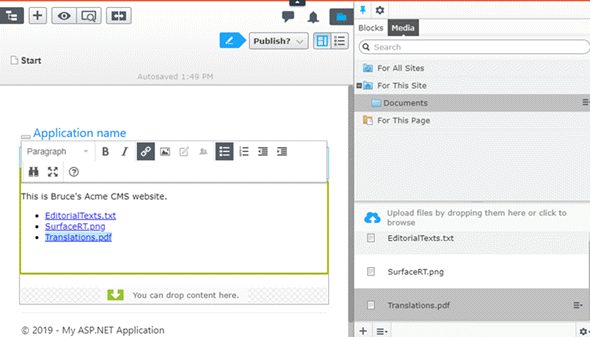


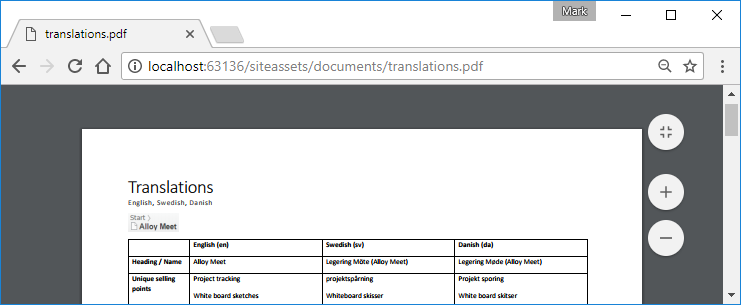
* If the **Blocks** tab is selected, you will not be able to drag and drop files, but it doesn’t tell you why.

1. Leave the browser running, and in Visual Studio, in **Solution Explorer**, expand the **App\_Data** folder, toggle **Show All Files**, refresh to show the **blobs** folder, and note that each uploaded file has its own folder that matches a content GUID in the CMS database, and a file for a version, as shown in the following screenshot:



* The default BLOB provider uses the filesystem. You can configure alternative BLOB providers, for example, to use Microsoft Azure Blob Storage or Amazon Web Services S3.

1. Back in the browser, edit the **Start** page.
2. In the **Main body**, create a bulleted list, drag and drop the three files, and note that links to the media files are created, and there is no special handling for images, as shown in the following screenshot:
3. **Publish** the Start page.
4. Switch to **Live** view, and click on each link to see the effect for visitors, as shown in the following screenshot of the PDF file:



## Customize handling of images by defining an image type

Before uploading image files, a little code is needed for the system to be able to recognize images as a special type of media and therefore customize how they are handled when dragged and dropped into a rich text property.

* The previously uploaded image, SurfaceRT.png, will remain registered as an “Any File” content type. To give it the special image handling, it would have to be deleted and uploaded again. There is a built-in Admin view feature to convert pages, but it cannot be used to convert other content types.

1. In Solution Explorer, expand **Models**, right-click **Media**, and click **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In the **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Media Type**, enter **ImageFile.cs** for the name, and click **Add**.
3. Modify the class to inherit from ImageData.
4. Change the DisplayName to **Image File**.
5. Add a Description of “Use this to upload image files.”
6. Uncomment the MediaDescriptor attribute, use the quick actions icon to fix the missing namespace and set the file extensions to: **png**,**gif**,**jpg**,**jpeg**
7. Uncomment the Description property and delete its Display attribute.

Your code should look something like the following:

using EPiServer.Core;

using EPiServer.DataAbstraction;

using EPiServer.DataAnnotations;

using EPiServer.Framework.DataAnnotations;

using System.ComponentModel.DataAnnotations;

namespace DemoSite.Models.Media

{

[ContentType(DisplayName = "Image File",

// your code will have a GUID

Description = "Use this to upload image files.")]

[MediaDescriptor(ExtensionString = "png,gif,jpg,jpeg")]

public class ImageFile : ImageData

{

[CultureSpecific]

[Editable(true)]

public virtual string Description { get; set; }

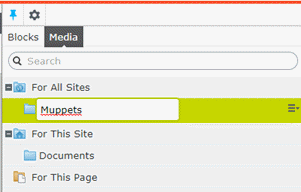
}

}

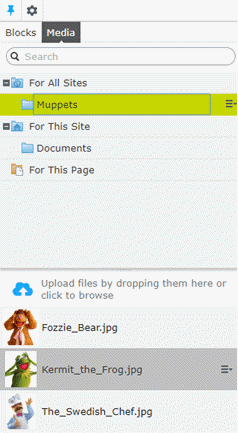
* Make sure you inherit from ImageData, not MediaData!

## Uploading images

1. Start the site, navigate to **episerver/cms** and log in as a CMS Admin.
2. Navigate to **CMS** | **Edit**.
3. In **Assets** pane, click **Media**, and create a folder named **Muppets** in **For All Sites**, as shown in the following screenshot:

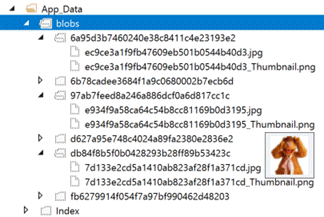


1. Upload images of the three Muppets from **solution source code project** in the folder **\Assets\Muppets\** into the **Muppets** folder, as shown in the following screenshot:



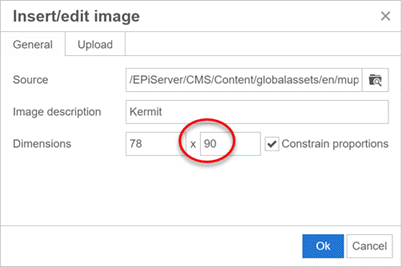
* One feature of inheriting from ImageData instead of MediaData is the automatic thumbnail generation.

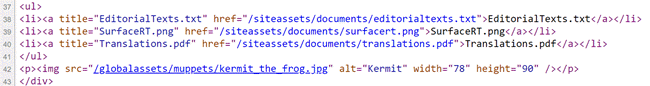
1. Leave the browser running, and in Visual Studio, in Solution Explorer, in the **App\_Data** folder, refresh the **blobs** folder, and note that each uploaded product image has its own folder that matches a content GUID in the CMS database, and a file for a version, and a file for a thumbnail, as shown in the following screenshot:



1. Back in the browser, double-click each image in the **Assets** pane, switch to **All Properties** view, edit the **Name** and **Description** properties, and then **Publish** them:
   1. **Fozzie\_Bear.png**:  
      Name: **Fozzie**, Description: **Fozzie the bear**.
   2. **Kermit\_the\_Frog.png**:  
      Name: **Kermit**, Description: **Kermit the Frog**.
   3. **The\_Swedish\_Chef.png**:  
      Name: **Swedish Chef**, Description: **The Swedish Chef**.

* Make sure that you publish the changes to the image metadata, or in a later exercise you will not see the names and descriptions.

1. Edit the **Start** page.
2. Drag and drop **Kermit the Frog’s** image into the **Main body** and note the image renders as an **<img>** element instead of a clickable hyperlink.
3. Select the image.
4. In the toolbar, click **Insert/edit image**, and change the height to **90**, as shown in the screenshot:
5. Publish the **Start** page.
6. Switch to **Live view**, to view the page as a visitor.
7. Right-click the page and choose **View page source**.
8. Note the HTML generated for visitors, especially the URLs for the documents and images that you uploaded as media assets, as shown in the following screenshot:



* /siteassets/ is the **For This Site** folder. /globalassets/ is the **For All Sites** folder.

## Inheriting Content Types

1. In Solution Explorer, expand **Models**, right-click **Pages**, and click **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Type**, enter **BioPage.cs** for the name, and click **Add**.
3. In the [ContentType] attribute, change the **DisplayName** to **Bio Page**.
4. In the [ContentType] attribute, set the description to **“Bio page for Acme employees”**.
5. In the [ContentType] attribute, add a **GroupName** attribute and set it to **“Main Content”**.
6. Chang the **BioPage** class inheritance to BioPage : StartPage
7. Remove the commented-out property and in its place, add the following property snippet: public virtual IList<string> Accomplishments { get; set; }
8. The final page template code should look something like the following (note that the **GUID** attribute value will be different for your code):

[ContentType(DisplayName = "Bio Page", GUID = "4294d990-3092-4dd6-bc20-b81235d25911",

Description = "Bio page for Acme employees", GroupName = "Main Content")]

public class BioPage : StartPage

{

public virtual IList<string> Accomplishments { get; set; }

}

1. Run the site, navigate to **episerver/cms** and log in as **Admin**.
2. In the site navigation panel, click the “hamburger” icon next to the **Start** page and choose the **New Page** option:  
   
3. Choose the **Bio Page** from the **Main Content** group and name it “**Swedish Chef**” and click **OK** to complete creating the new page instance from the content type.
4. Change the heading to **Swedish Chef**, add text to the **Main Body** such as “The Swedish Chef is a rock star gourmet chef” and add the **Swedish Chef** image from the **Media | For All Sites | Muppets** folder below the text in the **Main Body** area. The result should look close to the following image:  
   

**Remarks:** Note that you get the on-page editing experience even though you have not yet created a view template for the new content type. This is because the inherited parent, the StartPage, has a template that the CMS engine uses if the content type doesn’t have its own.

1. Use the **Publish** button to publish the chances so the page will be visible to visitors.
2. Switch the edit view **All Properties** and add the following, or something similar, to the **Accomplishments** list:
   1. Graduated from Le Cordon Bleu
   2. Taught Gordon Ramsay to bake
   3. Named most famous Muppet chef by USA Today
3. **Publish** the changes.
4. Switch to **View mode** by clicking the “globe” icon in the upper right of the page and note that the **Accomplishments** are not currently showing in the view.

**Remarks:** In order to view the **Accomplishments,** you’ll need to create a custom view template for the **BioPage** content type.

1. In Solution Explorer, expand **Controllers**, and right-click and choose **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Page Controller (MVC)**, enter **BioPageController.cs** for the name, and click **Add.**
3. Use the Quick Actions icon menu to add the missing namespace for the **BioPage** class.
4. Right-click inside the **Index** method body and choose the **Add View…** menu option.
5. In the **Add View** dialog select **Empty** for the **Template** drop-down and select **BioPage** for the **Model class**, leave everything else at the default and click the **Add** button.
6. Modify the code in the new **Index.cshtml** file to the following snippet:

<h2 @Html.EditAttributes(m => m.Heading)>

@(Model.Heading ?? Model.Name)

</h2>

<div class="container">

<div class="row">

<div class="col-8">

@Html.PropertyFor(m => m.MainBody)

</div>

<div class="col-4">

<ul>

@foreach(var item in Model.Accomplishments)

{

<li>@item</li>

}

</ul>

</div>

</div>

<div class="row">

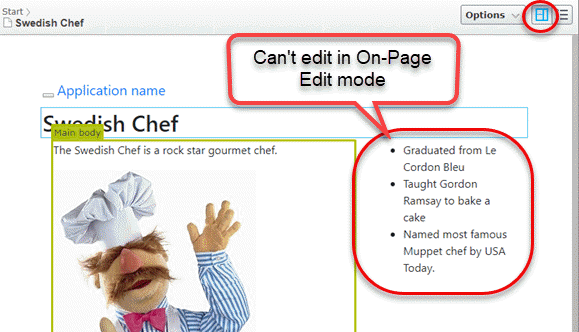
<div class="col-12">

@Html.PropertyFor(m => m.MainContentArea)

</div>

</div>

</div>

1. Run the site and manually navigate to **/episerver/cms**, log in as **Admin**.
2. In the navigation panel, click to select the **Swedish Chef** page, make sure you are in **On-Page Editing** mode and note that the **Accomplishments** list doesn’t have an editing panel like the other properties and you cannot edit the list in this mode.  
   

**Remarks:** If you switch to **All Properties** view then you would be able to edit the list. To enable the ability to edit properties displayed as an unordered list in On-Page Editing view, there are a couple ways you can do it. The first quick way is to just add a call to an HTML helper method provided by the Episerver framework in the containing **<ul>** element. The following steps will show this method.

1. Modify the **<ul>** element in the **Index.cshtml** file by adding **@Html.EditAttributes(m => m.Accomplishments)** inside the element, the complete **<ul>** should look like the following:

<ul @Html.EditAttributes(m => m.Accomplishments)>

@foreach(var item in Model.Accomplishments)

{

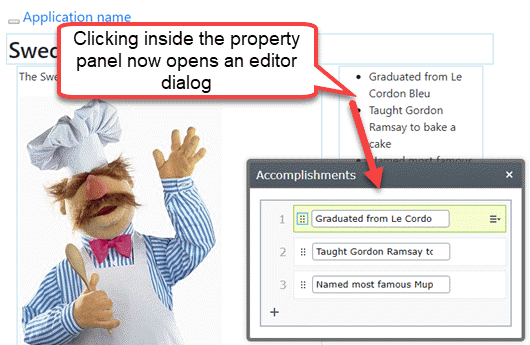
<li>@item</li>

}

</ul>

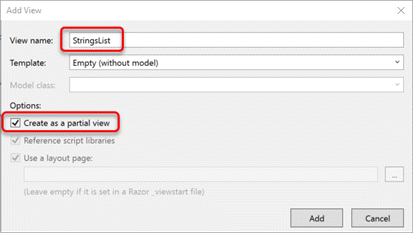
1. Save the changes to file in Visual Studio and then switch back to the browser window and refresh the page.

**Remarks:** You don’t have to re-run the site if the only thing you have changed is a **cshtml** file, you can just refresh the page in the browser.

1. Verify that clicking inside the **Accomplishments** panel in **On-Page Editing** mode opens and editor dialog as shown in the following image:  
   

**Remarks:** Another way to get the On-Page Editing for a property is to use the **@Html.PropertyFor** helper method. The problem with that is that the default rendering will not show the list as an unordered list but instead all the items will become just a single line of text. To use the **@Html.PropertyFor** and get the custom rendering you want, you can create a custom display template and then reference it with a **UIHint** on the property. The following section will show how to do that as an optional exercise.

### IList<string> Custom Display Template (Optional)

1. In Visual Studio, add a new folder named **DisplayTemplates** to the **Views | Shared** folder in Solution Explorer.
2. Right-click the new **DisplayTemplates** folder and choose **Add | View…** from the menu.
3. Name the new view **StringsList** and check the **Create as a partial view** checkbox, see the image below, and click the **Add** button.  
   
4. Add the following code to the **StringsList.cshtml** file:

@model IEnumerable<string>

@if (Model != null && Model.Any())

{

<ul>

@foreach (var item in Model)

{

<li>@item</li>

}

</ul>

}

1. Edit the page model file **BioPage.cs** and add [UIHint("StringsList")] to the top of the **Accomplishments** property, it should look like the following code snippet:

[UIHint("StringsList")]

public virtual IList<string> Accomplishments { get; set; }

**Remarks:** The **UIHint** needs to point to the name of the view stored within the **Views | Shared | DisplayTemplates** folder. Now you can use the **PropertyFor** Html helper method and it will use the partial view when rendering the property on the page but still give you editing ability in On-Page Edit mode. The next steps will test this.

1. Edit the **Views | BioPage | Index.cshtml** file and replace the whole **<ul>** block with: @Html.PropertyFor(m => m.Accomplishments)
2. Run the site and navigate to **episerver/cms**, log in as **Admin**.
3. In the navigation panel, click to select the **Swedish Chef** page and note that the **Accomplishments** list can now be edited in **On-Page Editing** mode and renders correctly as an unordered list.

**Remarks:** This method is arguably better for reusability. If you just needed it for a single view or you wanted a more custom rendering, then using the first method with the **EditAttributes** Html helper could be the way to go.

# Defining Blocks

## Creating an editorial block type

1. In Solution Explorer, expand **Models**, right-click **Blocks**, and click **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Type**, enter **EditorialBlock.cs** for the Name, and click **Add**.
3. Change the **DisplayName** to **Editorial**.
4. Add a **Description** of “Use this for a rich editorial text that will be reused in multiple places.”
5. Add an **XhtmlString** property named **MainBody**, your code should look something like the following:

[ContentType(DisplayName = "Editorial", GUID = "c4abefcb-bc4a-4986-b265-2448ee0b6dbe",

Description = "Use this for a rich editorial text that will be reused in multiple places.")]

public class EditorialBlock : BlockData

{

[Display(Name = "Main body",

Description = "The main body will be shown in the main content area of the page," +

" using the XHTML-editor you can insert for example text, images and tables.")]

public virtual XhtmlString MainBody { get; set; }

}

1. In Solution Explorer, expand **Controllers**, and right-click and choose **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.

**Remarks:** Note that you could create a “controller-less” Block by creating a view in the **Shared** folder with the same name as the Block class. In a later part of the exercise you’ll use the controller you just created to add some dynamic content into the view. If you didn’t need extra logic, controller-less blocks are more efficient than block templates with controllers, so you should use them whenever possible.

1. In **Add New Item** dialog, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Block Controller (MVC)**, enter **EditorialBlockController.cs** for the name, and click **Add.**
2. Use the Quick Actions lightbulb to fix the missing namespace for the **EditorialBlock** class.
3. Right-click inside the **Index** method body and choose the **Add View…** menu option.
4. In the **Add View** dialog select **Empty** for the **Template** drop-down and select **EditorialBlock** for the **Model class**, leave everything else at the default and click the **Add** button.
5. Add the following code snippet to the new **Index.cshtml** file:

<div class="block span">

@Html.PropertyFor(m => m.MainBody)

</div>

1. Run the site and create a folder named **Editorials** in the Assets pane Blocks and under **For This Site**.
2. Name the new block **Famous Quotes**. (If you only have a single block you won’t see the dialog that lets you choose)
3. Add some quotes and **publish**.

|  |
| --- |
| “Life is like a box of chocolates” – Gump  “Just Do It!” – Nike  “One Ring to rule them all” – Tolkien |

1. Point out that the yellow banner at the top of the page indicates if the block is used anywhere in the site.
2. Edit the **Start Page** and drag and drop the **Editorial** block into the **Main Content Area** and publish.
3. Go back to the edit view of the Editorial block and point out how the banner now shows the message “**Changes made here will affect at least 1 item**”.

# Dependency Injection

## Out-of-the-box StructureMap

Out-of-the-box CMS uses StructureMap for dependency injection and service location. This next demo will make use of service location to get an instance of **IContentRepository** that will in turn be used to retrieve references to all child pages from the root page. You’ll use this to build a basic navigation menu on the shared **\_Layout.cshtml** view template.

1. Edit the **Views | Shared | \_Layout.cshtml** view template and add the following code to the top of the file:

@using EPiServer.ServiceLocation

@using EPiServer.Core

@using EPiServer

@{

// Get a reference to the IContentRepository

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

// Get the child pages of the start page.

var childPages = cr.GetChildren<PageData>(ContentReference.StartPage);

}

**Remarks:** Here you’re using the out-of-the-box **ServiceLocator** to retrieve a reference to whatever is the currently registered **IContentRepository** service using the static **GetInstance** method. With the reference to the content repository service you are then calling the **GetChildren** method passing in a static reference to the page that has been configured as the sites **StartPage** that returns back an **IEnumerable** of **PageData** objects. Note that **PageData** is a base class for any page types you create. Next, we’ll modify the html menu, created by the Visual Studio scaffolding feature that created the **\_Layout.cshtml** file, to display the start page and any child pages.

Note that this could be done more elegantly using some covariance along with a base controller, but this method is quick and easy for demo purposes.

1. Modify the **div** element that defines the navigation so that it expands to show all the items by adding **navbar-expand**, it should look like the following:

<div class="navbar navbar-expand navbar-inverse navbar-fixed-top">

1. Inside the **div** **class=”navbar-header”**, replace the **@Html.ActionLink** helper method call line with the following:

@Html.PageLink("Home", ContentReference.StartPage, null, new { @class = "navbar-brand" })

1. Add the following **foreach** loop inside the existing **ul** element:

<ul class="nav navbar-nav">

@foreach (PageData page in childPages)

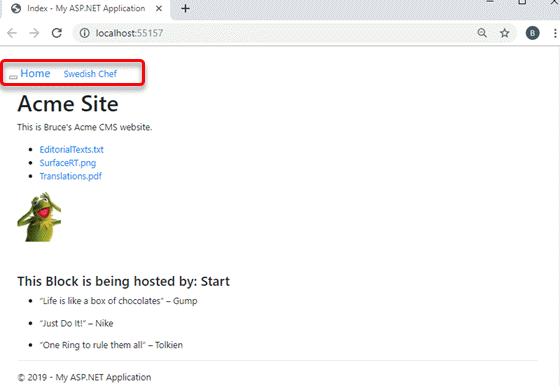
{

<li>@Html.ContentLink(page.ContentLink, null, new { @class = "nav-item nav-link" })</li>

}

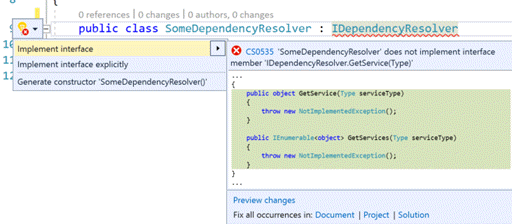
</ul>

1. Run the site and verify that it looks something like the following image and you can navigate to the **Swedish Chef** page successfully.



## Setting up MVC Controller Dependency Injection

To enable controllers to have constructors with parameters to inject dependencies you must register a dependency resolver with ASP.NET MVC. In the following steps, you’ll create an initialization module to make sure that Episerver’s integration with StructureMap is registered as the resolver when the website starts up.

1. In Visual Studio and the site project, right-click the **Business** folder and add a new folder named **SiteDependencyResolver**.
2. Right-click the new **SiteDependencyResolver** folder and choose **Add** | **Class**… and name the new class **ServiceLocatorDependencyResolver.cs**, click **Add** complete creating it.
3. Add the **IDependencyResolver** interface to the class declaration and use the **Quick Actions** icon to fix the missing namespace.
4. Use the **Quick Actions** icon again and choose **Implement interface** as shown in the following image:  
   
5. Replace the code in the body of the class with the following and fix any missing namespaces:

readonly IServiceLocator \_serviceLocator;

public ServiceLocatorDependencyResolver(IServiceLocator serviceLocator)

{

\_serviceLocator = serviceLocator;

}

public object GetService(Type serviceType)

{

if (serviceType.IsInterface || serviceType.IsAbstract)

{

return GetInterfaceService(serviceType);

}

return GetConcreteService(serviceType);

}

private object GetConcreteService(Type serviceType)

{

try

{

// Can't use TryGetInstance here because it won’t create concrete types

return \_serviceLocator.GetInstance(serviceType);

}

catch (ActivationException)

{

return null;

}

}

private object GetInterfaceService(Type serviceType)

{

object instance;

return \_serviceLocator.TryGetExistingInstance(serviceType, out instance) ? instance : null;

}

public IEnumerable<object> GetServices(Type serviceType)

{

return \_serviceLocator.GetAllInstances(serviceType).Cast<object>();

}

**Remarks:** By inheriting **IDependencyResolver** the MVC engine will be able to instantiate services defined within a controller’s constructor. The **IServiceLocator** interface comes from Episerver framework and it is what will be used to retrieve individual instances of services or a list based on the service type passed in. The next thing to do is to register this **IDependencyResolver** with the MVC engine so it will know to use it.

1. In Visual Studio’s Solution Explorer, expand the **Business** folder, right-click the **Initialization** folder, and choose **Add** | **New Item**…, or press *Ctrl* + *Shift* + *A*.
2. In **Add New Item** dialog window, navigate to **Installed** | **Visual C#** | **Episerver**, choose **Initialization Module**, enter **RegisterDependencyResolverInitializationModule.cs** for the name, and click **Add**.
3. **Important:** Remove the ModuleDependency attribute from the top of the class declaration.

**Remarks:** Code that registers the dependency resolver for MVC must be done after the IOC configuration is done (that is it can’t be done in ConfigureContainer method) but it also needs to be done before a certain MVC initialization module in CMS is executed (meaning the **IInitializableModule** that register dependency resolver should not use **ModuleDependency** attribute).

1. Modify the **Initialize** method with the following code and fix any missing namespaces:

DependencyResolver.SetResolver(new ServiceLocatorDependencyResolver(context.Locate.Advanced));

**Remarks:** The next thing to do is to test the ability of the dependency resolver to create instances of services through a controller’s constructor parameter list.

1. Edit the **Controllers | EditorialBlockController.cs** file and add the following constructor and field to the class definition and fix any missing namespaces:

private IPageRouteHelper \_pageRouteHelper;

public EditorialBlockController(IPageRouteHelper pageRouteHelper)

{

\_pageRouteHelper = pageRouteHelper;

}

1. Add the following line of code into the **Index** method before the **return**:

//Using ViewBag for convenience, a better approach would

//be to create and use a view model.

ViewBag.pageName = \_pageRouteHelper.Page.Name;

**Remarks:** As the comment says, creating and using a view model would be a more elegant approach but for demonstration purposes, the ViewBag will work. Note that the **IPageRouteHelper** gives you access to the current page that the Block instance is hosted on. It’s typed as **PageData** but you could cast it to something more specific if you needed access to properties defined by the more specific type.

Now you need to modify the Razor view template to display the results.

1. Edit the **Views | EditorialBlock | Index.cshtml** file and add the following code above the current **div** element:

<h4>This Block is being hosted by: @ViewBag.pageName</h4>

1. Run the site and note that any instance of the editorial block shows the name of the page that it has been added to.

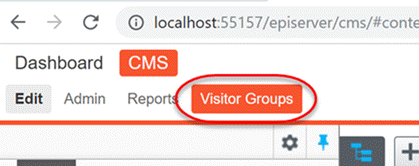
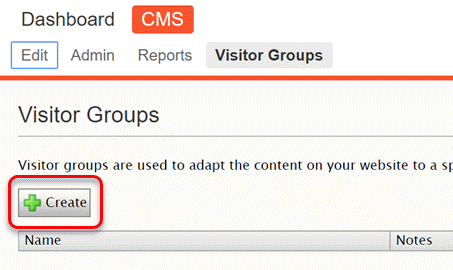
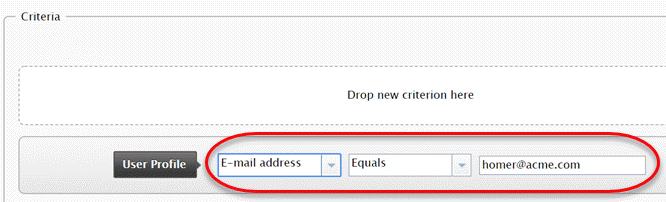
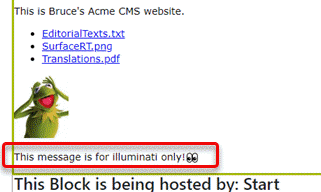
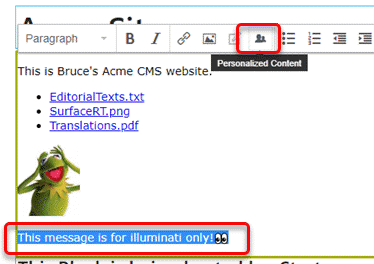
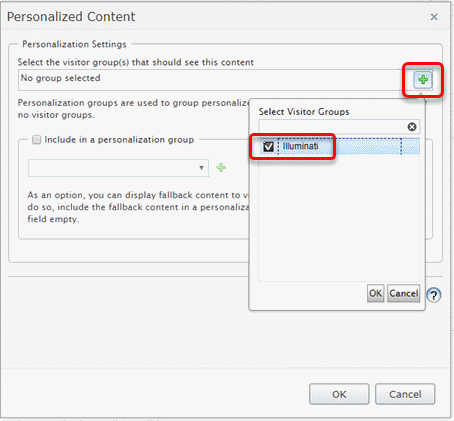
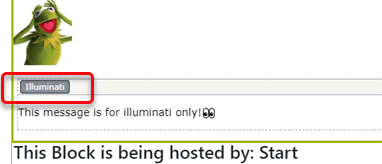
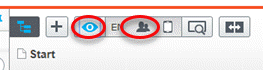
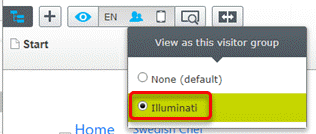
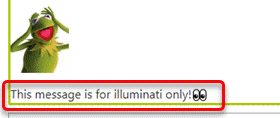


**Remarks:** Feel free to further test by adding the Editorial Block to the **Swedish Chef** page, the heading should reflect the name of the page correctly.

# Personalization

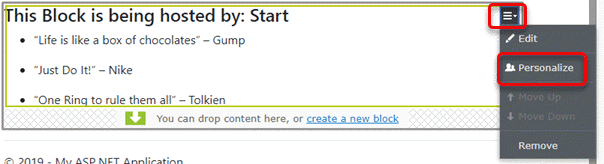
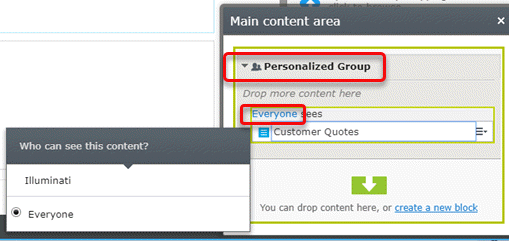
## Setting up a Visitor Group

Using visitor groups for personalizing content is an easy and effective method but it does require manual configuration. The following exercise will walk-through creating a visitor group and using it to personalize content.

1. Run the demo site and manually navigate to **/episerver/cms**, login as Admin.
2. Click the **Visitor Groups** link on the top navigation:  
   
3. Click the **Create** button on the **Visitor Groups** view:  
   
4. Name the group **Illuminati** and then note the various **Criterion** you can choose from on the right-hand side of the view.
5. Drag and drop the **Site Criteria | User Profile** into to the **Criteria** panel and set the values to **E-mail address | Equals |** [**homer@acme.com**](mailto:homer@acme.com) and click the **Save** button.  
   
6. Click the **Edit** link in the top menu and select the **Start** page for editing if it is not already selected.
7. Enter text into the **MainBody** property of the **Start** page that you want only members of the **Illuminati** visitor group to see, something like the following screen image:  
   
8. Highlight the text you typed in the previous step and then click the **Personalized Content** icon from the TinyMCE toolbar:  
   
9. Click the **+** icon at the top of the **Personalization Settings** dialog and then click the **Illuminati** visitor group to select it, click the **OK** button on the dialog and then click the second **OK** button to save and close the main dialog:  
   
10. Note that while you are editing the **Main Body** property you can see the personalized text with labels for the personalization groups that see it. If you click outside the property editor, the text will disappear since the Admin account is not currently a member of **Illuminati**:  
    
11. **Publish** the page and then click the **View Mode** (globe icon) to see how the page is rendered.
12. Note that you don’t see the text because Admin is not yet a member of the personalization group. Use the browser back button to navigate back to the CMS edit view.
13. Click the **Toggle view settings** icon and then click the **Select to view as a visitor group** icon:  
    
14. Select **Illuminati** from the **View as this visitor group** dropdown:  
    
15. Note how you can now see the Illuminati message in on-page-edit mode without having to be in the TinyMCE edit mode:  
    

**Remarks:** The **Toggle view settings** only affects the **on-page-edit** view and **preview mode**, if you switch to **View Mode** it will be based on who you are logged in as. You could test that by creating a user account with the [**homer@acme.com**](mailto:homer@acme.com) email and logging in as the user.

You can also use visitor groups to personalize content placed in a **ContentArea** property. The following steps show how this is done.

1. Click the hamburger icon on the **Customer Quotes** block and click the **Personalize** option:  
   
2. Note that now the block is nested in a **Personalized Group** region that you can drag additional items into and set who can see them:  
   

**Remarks:** Each item dropped into a **Personalized Group** can have unique personalization settings, this includes media types (with a view template) and partial pages. You can also drag and drop content outside the personalization group.

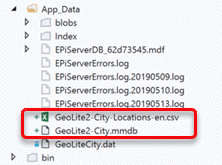
1. Either publish or revert the changes, whichever you prefer.

## Adding Geographic Visitor Group Criteria

Visitor group criteria is extensible and one of the easiest ways to do that is by downloading NuGet package add-ons. In the following steps you’ll download one that supports grouping be geographic information.

1. In Visual Studio’s Solution Explorer, right-click the project and select **Manage NuGet Packages…** from the menu.
2. Make sure the **Package source** is set to either **All** or **Episerver NuGets** and type **episerver.personalization** in the **search** field of the **Browse** tab.
3. Select the **EPiServer.Personalization.MaxMindGeolocation** package and click the **Install** button to install the package and whatever dependencies it requires.

**Remarks:** The package installs assemblies with classes for accessing a database named **GeoLite2-City** that is provided and maintained by MaxMind. The database and a supporting CSV file are not included in the package and must be downloaded separately.

1. To get the database and necessary CSV file, navigate to <https://dev.maxmind.com/geoip/geoip2/geolite2/> and download the GeoLite2 City MaxMind DB and Locations CSV files.
2. Extract the zipped files and then drag the **GeoLite2-City.mmdb** and **GeoLite2-City-Locations-en.csv** files and drop them onto the **App\_Data** folder in Visual Studio’s Solution Explorer so they are copied and include with the project:  
   

**Remarks:** You may notice there was already a file in **App\_Data** named **GeoLiteCity.dat**. Previous versions of CMS included built-in support for MaxMind’s GeoLite database. That database is no longer supported but is still include for backwards compatibility. The new database, GeoLite2, is accessed with classes that come from the NuGet package installed in the previous steps.

1. Open **Web.config**, add a <geolocation> element inside the <episerver.framework> element, as shown in the following markup, and save changes:

<geolocation defaultProvider="maxmind2">

<providers>

<add name="maxmind2"

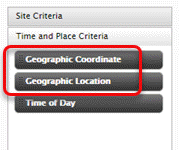
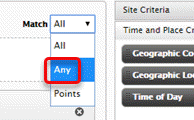
type="EPiServer.Personalization.MaxMindGeolocationProvider, EPiServer.Personalization.MaxMindGeolocation"

databaseFileName="App\_Data\GeoLite2-City.mmdb"

locationsFileName="App\_Data\GeoLite2-City-Locations-en.csv" />

</providers>

</geolocation>

1. Start the site and manually navigate to **/episerver/cms** and log in as **Admin**.
2. Click the **Visitor Groups** link in the top navigation.
3. Click the **Illuminati** visitor group to open and edit it.
4. Click the **Time and Place Criteria** tab in the panel and note there are two new criterion, **Geographic Coordinate** and **Geographic Location**:  
   
5. Drag the **Geographic Location** criteria to the **Illuminati** group criteria panel and se the values to your location, In my case that’s **Continent: North America**, **Country: United States** and **Region: Florida**:  
   
6. Change the criteria match drop-down to **Any** and click **Save** to save the changes:  
   

**Remarks:** The **Geographic Coordinate** criteria uses Google maps to allow you to select a point and radius distance from that point. For that to work you need an account key with Google that you add in with the URL to the **episerver ->** **applicationSettings** element as the attribute, googleMapsApiV3Url="https://maps.google.com/maps/api/js?key=<<you key goes here>>", of the **Web.config** file. The **applicationSettings** element should already be there, you just add the new attribute and value.

1. Click the “globe” icon to switch to live view and note that you still **don’t** see the Illuminati message.

**Remarks:** The geographic criteria looks at the IP address in the incoming client request and compares that against the **GeoLite2** database using a built-in service instance of **IClientIPAddressResolver**. The problem with the current demo is that I’m accessing the site using **Localhost** which doesn’t get a valid IP address in the request header.

To solve this problem without having to change from using Localhost, the following steps are going to show how to create a custom service that will replace the default. This technique can be used with just about any service you get out-of-the-box with Episerver. In this case you’ll be creating a custom instance of **IClientIPAddressResolver**.

### Getting Geographic Criteria to work with Localhost

1. In Visual Studio’s Solution Explorer, right-click the **Business** folder and choose **Add | Class…**
2. Name the class **CustomIPAddressResolver**.
3. Set the class to implement **IClientIPAddressResolver** and use the **Quick Actions** icon to fix the missing namespace:public class CustomIPAddressResolver : IClientIPAddressResolver
4. Use the **Quick Actions** icon a second time to **Implement interface**.
5. Add the following code inside the body of the **ResolveAddress** method:

IPAddress pAddress;

if (httpContext.Request.UserHostAddress.Equals("::1")) // Website is running localhost so define hardcoded a valid IP address

{

return IPAddress.Parse("73.118.88.1");

}

else if (IPAddress.TryParse(httpContext.Request.UserHostAddress, out pAddress))

{

return pAddress;

}

return IPAddress.None;

**Remarks:** With the site being accessed with **Localhost** the IP address that the client registers in the request header is “**::**1”, this code is looking for that and replacing it with a valid public IP. Note that the IP address, 73.118.88.1, is from my Internet provider and is located in Florida. To get the geographic location to work for your region you’ll need to find a public IP from your area. Either that or change the geographic location criteria to Florida and use the address provided in the code.

The last thing needed to make this work is to replace the out-of-the-box service with this custom implementation of **IClientIPAddressProvider**. Since we have an IOC system in place for initializing services we just need to use it for our custom service.

1. In Visual Studio’s Solution Explorer, right-click the **Business | Initialization** folder and choose **Add | New Item…**.
2. From the **Episerver** templates choose the **Initialization Module** template and name it **IPResolverInitializationModule.cs**.
3. Change the interface declaration from **IInitializableModule** to **IConfigurableModule** and use the **Quick Action** icon to fix the missing namespace.
4. Use the **Quick Action** a second time to implement the interface, adding the missing **ConfigureContainer** method.
5. Add the following code into the **ConfigureContainer** method:

context.Services.Intercept<IClientIPAddressResolver>((locator, defaultResolver) => new CustomIPAddressResolver());

**Remarks:** The **Intercept** method allows you to replace any default services with custom implementations. Note that you can create an overloaded constructor for your custom service and pass the default implementation in so that you can reuse any logic from the default you like. In our **IClientIPAddressResolver** example that’s not necessary.

1. Run the site and you should now see the Illuminati text. If you don’t, you might need to adjust the region value to reflect the public IP you used.