Developer 101 Training

August 2014

AGILITY

Table of Contents

Developer 101 Training

Introduction

What is Agility?

Let's start with the most basic information. Agility is a SaaS based software platform that is hosted in Microsoft Azure. It is built from the ground up in ASP.NET and allows you to build websites, and mobile sites, as well power content to mobile applications including a variety of social platforms. For websites and mobile sites, Agility provides an ASP.NET Framework that can be used with either Web Forms or MVC. For these as well as the other platforms Agility provides a number of API layers in order to make integration with external systems a breeze.

Agility includes the Agility Content Manager which is the online tool to log into and manage your content. It also includes a Content Server which is responsible for publishing content to the various digital channels. This is accessed through the Agility.Web.Dll or through the direct API's.

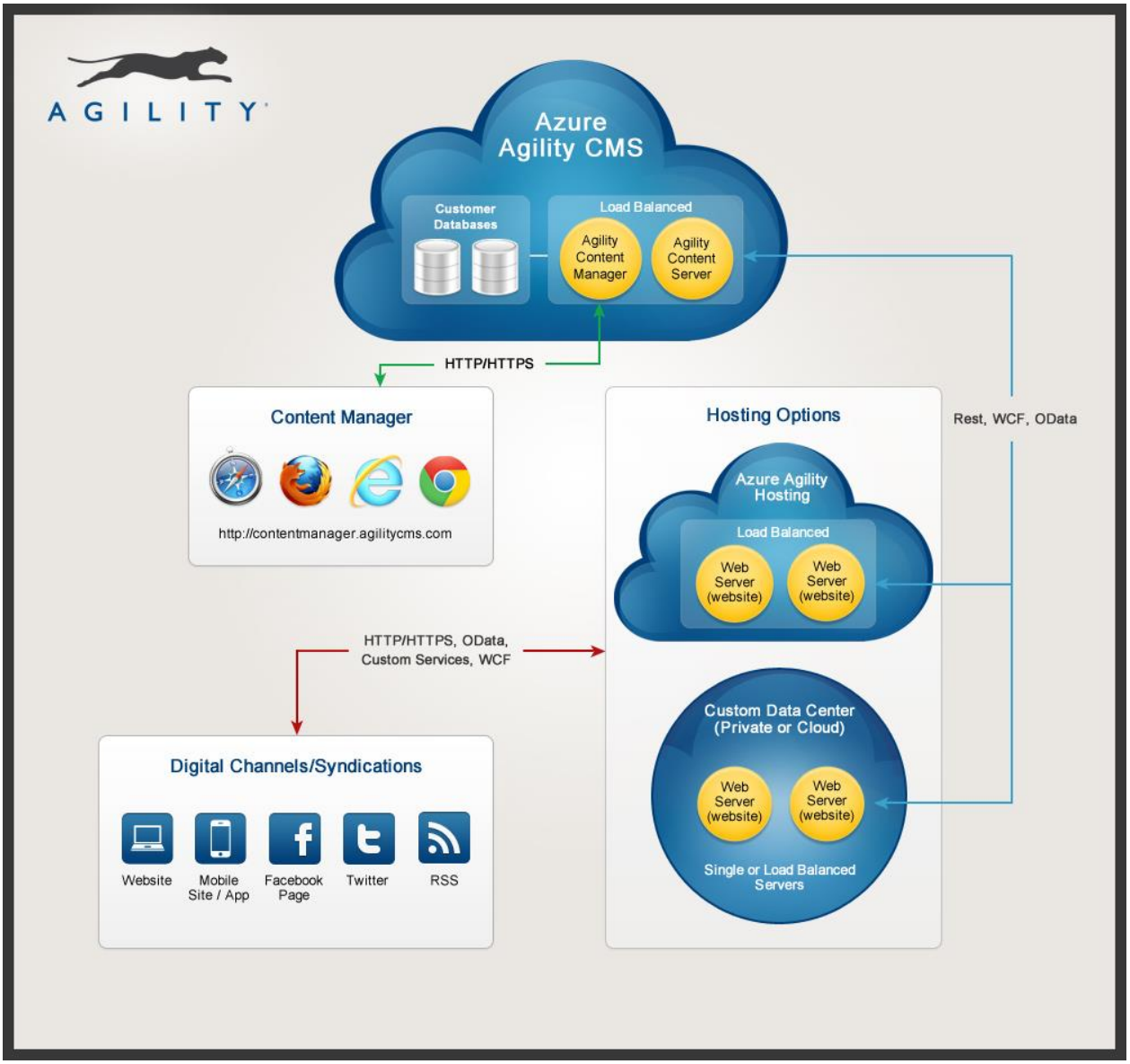
Agility also includes a Web Content Management (WCM) Component and a User Generated Content (UGC) Component. WCM specializes in publishing content out to various digital channels, while UGC specializes in retrieving and moderating content that comes in from the websites including website profiles and other social media content.

Why Developers Love It

- Cloud CMS – no software or databases to install, upgrade, or manage
- No proprietary templating languages to learn
- Unobtrusive development – build your site the way you want
- Customize your content architecture by creating content and module definitions that suit the needs of your content
- Use built-in controls for content entry such as WYSIWYG Editor, File Picker, Image, Image Gallery, CheckboxLists, Dropdowns, SearchListBox or use a Custom Input Form and hook into the Input Form API for events, content, or integrate with other external APIs, all using JavaScript
- Supports both ASP.NET MVC and ASP.NET Web Forms websites
- REST and OData APIs available to integrate your content into any platform such as apps, social media, or anything else you can dream up
- Optionally manage site CSS, JS, and HTML templates on-the-fly within the browser, using Inline Code

- Flexible site hosting options - Host your site with us in the Azure Cloud or host it yourself
- Built with performance in mind – Agility fully supports load balanced web servers, output caching and donut caching
- User Generated Content (UGC) API to manage content submitted through your website such as Form Submissions, Comments, and Website Users
- Media and Documents delivered via CDN
- Auto-generated Strongly Typed C# API for all your module and content definitions

Agility Architecture



Azure Agility CMS

Agility CMS is hosted in Azure: Microsoft's Cloud Platform and managed 100% by the Agility product team. This consists of the following components:

- **Customer Databases**: Where all of your content is stored securely in a scalable environment
- **Agility Content Server**: The server responsible for synchronizing content to your websites

- **Agility Content Manager**: Serves as a central point for configuring and managing content, as well as managing servers to which content is synchronized and other advanced settings specific to your Agility instance

Hosting Options

We offer load-balanced hosting in our Azure environment, however you may optionally use a third-party private or cloud data center.

If you are hosting with us, you will have a Stage and Live server. If you are hosting on your own, it's still highly recommended that you have a Stage and Live server so you can test your deployment on Stage before making any changes to Live.

Content Synchronization

Publishing content in the Content Manager triggers the synchronization process, which sends a message to each of your configured web servers instructing them to contact the Agility content service to pull the latest content (as a delta). When each web server pulls content down, it stores it in its Content directory as serialized bin files. This means that your website will serve content directly from its own physical disk, rather than fetching over the web.

Website Environments

An Agility website can be placed in one of two modes which can be controlled via the web.config:

- **Development Mode**: Pulls the latest content (staging and published content) in real-time
- **Production Mode:** When not in Development Mode, an Agility website can be viewed in a Preview Mode (only invoked by a secret hash passed into querystrings for internal use) or Live Mode (public facing website).
    - **Preview Mode**: Pulls the latest content (staging and published content) in real-time
    - **Live Mode:** Content is pushed to each web server on publish, Pushed when content is published and Agility does a content sync.

A typical Agility website will have a Stage and a Live server. Both Stage and Live should both NOT be in Development Mode. The idea here is to have two nearly identical servers so you can test your code changes in a simulated live environment before deploying to Live.

Development Workflow

A typical development workflow will look something like this:

1. Running your site locally in Visual Studio in Development Mode, you make changes to the code base. You test your changes in your local browser.

2. Once satisfied with your changes, you will deploy the website to your Stage server (in Production Mode) usually via FTP.
3. You test your changes again on Stage to verify they are working the way you want. You can test the Stage server in Preview Mode and Live Mode.
4. Once this has passed QA, you deploy your changes to your Live server usually via FTP.

Source Control

We highly recommend using some sort of Source Control that integrates with Visual Studio whether it be Team Foundation Server or Git Hub. Part of maintaining a website is maintaining its code base even if it's just one user.

Microsoft has a SaaS solution called Visual Studio Online which allows you to host your code in the cloud – and is Free for up to 5 users. See http://www.visualstudio.com/en-us/products/what-is-visual-studio-online-vs.aspx for more information.

Content States

All content in Agility has a State which will influence what appears on the live website, on preview, and what won't appear at all. There are 3 States:

- **Staging**: All new or edited content is placed in a Staging State. All content in Staging will be visible on Preview – it will not appear on the Live site unless it has been changed to a Published State. Published content which has been edited will stay in Staging until the latest changes are published. Staging includes several Sub-States designed for internal approval workflows and content scheduling.
    - **Awaiting Approval**: Types of content that has been previously defined to require approval needs to be approved before it can be Published
    - **Approved:** Content that has been approved, but not yet Published
    - **Scheduled for Release**: Content that has been scheduled to be automatically Published at a given date and time
- **Published:** Content has been synchronized and is available to the live website
- **Unpublished**: Content that has been either explicitly removed from the website either on demand or via Content Scheduling. Unpublished content will not appear in Preview or Live.
    - **Expired:** Content that has been scheduled to come down after a specific date and time

Connecting to your Content

ASP.NET MVC sites connect to Agility by utilizing several components that must be included in the website project.

1. **Agility.Web section in Web.Config** – This contains options to switch on/off Development Mode, ContentCacheFilePath, Site Name and SecurityKey (for authentication) and setting the TraceLevel and LogFilePath for error logging.

   Agility specific Web.Config sections are shown below:

```
<configuration>
  <configSections>
    <!-- Agility.Web Config Group -->
    <sectionGroup name="agility.web">
      <section name="settings" type="Agility.Web.Configuration.Settings, Agility.Web" allowLocation="true"
allowDefinition="Everywhere" restartOnExternalChanges="false" requirePermission="false" />
    </sectionGroup>
  </configSections>
  <!--Agility  settings-->
  <agility.web>
    <settings applicationName="My Site" developmentMode="true" contentCacheFilePath="c:\AgilityContent\">
      <websites>
        <add websiteName="My Site" securityKey="{securityKeyGoesHere}" />
      </websites>
      <trace traceLevel="Warning" logFilePath="c:\AgilityLogs\SampleMVC4.log" emailErrors="false" />
    </settings>
  </agility.web>


  <system.web>
    <siteMap defaultProvider="AgilitySiteMapProvider">
      <providers>
        <add name="AgilitySiteMapProvider" type="Agility.Web.Providers.AgilitySiteMapProvider,
Agility.Web" />
      </providers>
    </siteMap>
  </system.web>

  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />
    <services>
      <service behaviorConfiguration="agilityWebsiteServiceBehavior"
name="Agility.Web.AgilityWebsiteService">
        <endpoint binding="wsHttpBinding" bindingConfiguration="agilityWebsiteServiceBinding"
contract="Agility.Web.IAgilityWebsiteService" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="agilityWebsiteServiceBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceMetadata httpGetEnabled="true" httpGetUrl="" />
        </behavior>
        <behavior name="">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <wsHttpBinding>
        <binding name="agilityWebsiteServiceBinding" receiveTimeout="00:30:00"
maxReceivedMessageSize="2147483647">
          <security mode="None">
            <transport clientCredentialType="None" />
            <message establishSecurityContext="false" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

2. **AgilityWebsiteService.svc** – This is the service that is used to synchronize and pull down content from Agility. This file can be provided to you on demand.
3. **Agility.Web.dll** – This is the DLL which is used to connect to your Agility content via C#. The latest version can always be downloaded from **Developer Downloads** found in the Agility Content Manager.
4. **Agility Page Route in your RouteConfig / Global.asax** – This is required for rendering the default Agility page routes.

```
//Agility Builtin Route
routes.MapRoute("Agility", "{*sitemapPath}", new { controller = "Agility", action =
"RenderPage" },
    new { isAgilityPath = new Agility.Web.Mvc.AgilityRouteConstraint() });
```

We recommend viewing our Sample MVC site for details on how these are already set up and you can use it for reference if you need to create a new project.

Getting Started

What You'll Learn

This training will begin by covering the basics of getting started developing on Agility. We will primarily be focusing on Web Content Management and website deployment. The training has been broken down into a set of labs that will allow you to build on your knowledge as you move through the training. Once complete, you will have the initial base skills necessary to develop an Agility based website and deploy it into production.

What You'll Need

To complete the labs in this training course, you'll need to have a Sandbox Sample Site setup for you and have a copy of the site source code. Our current Sandbox Sample Site is a .NET 4.0 site using ASP.NET MVC 4.

You will need to have access to an Integrated Development Environment (IDE) such as Visual Studio. We recommend using Visual Studio 2013 – there is a free version Visual Studio Express for Web which is sufficient.

Finally, you'll need a web browser with an internet connection to access the Agility Content Manager. The Agility Content Manager supports all modern browsers.

How to Get Set Up

If you haven't done so already, contact our sales team to get a Sample Site set up. We'll send you a download link where you can get the source code for your sample site. Ensure you also have received your FTP credentials to where you can deploy code changes.

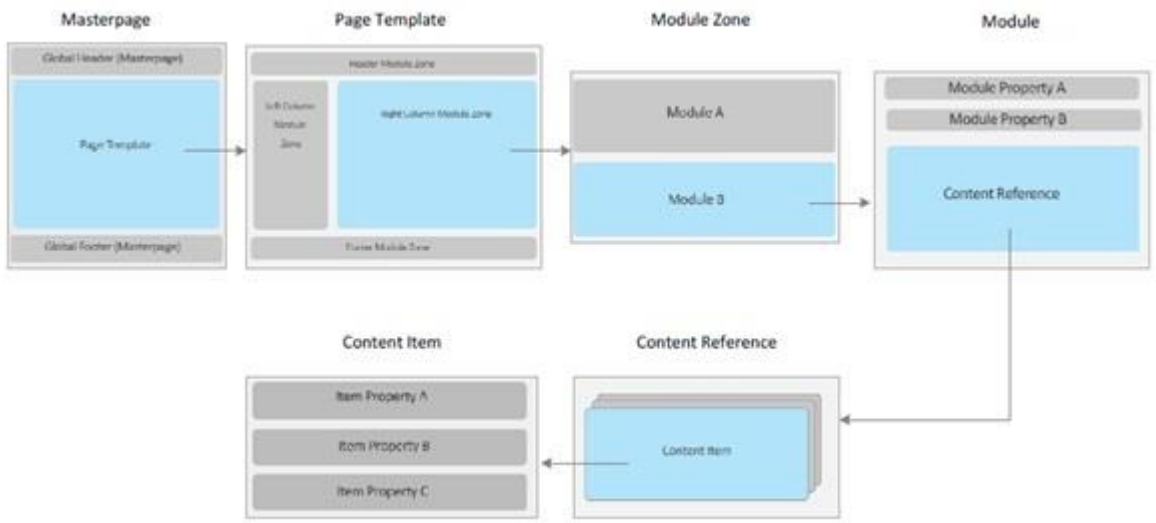
Once you've got the source files, select the .sln Solution File to open the project in Visual Studio. The site will automatically be configured to connect to your Sample Site instance and will be running in Development Mode using IIS Express. Click "Start Debugging" or "Start without Debugging" to run your site locally in your favourite web browser.

Site Walkthrough

Once you have the Sample Site running locally, click the "Start Tour" link found in the footer of the site. This will launch a guided tour taking you through some basics of how this site is put together.

Web Content Management

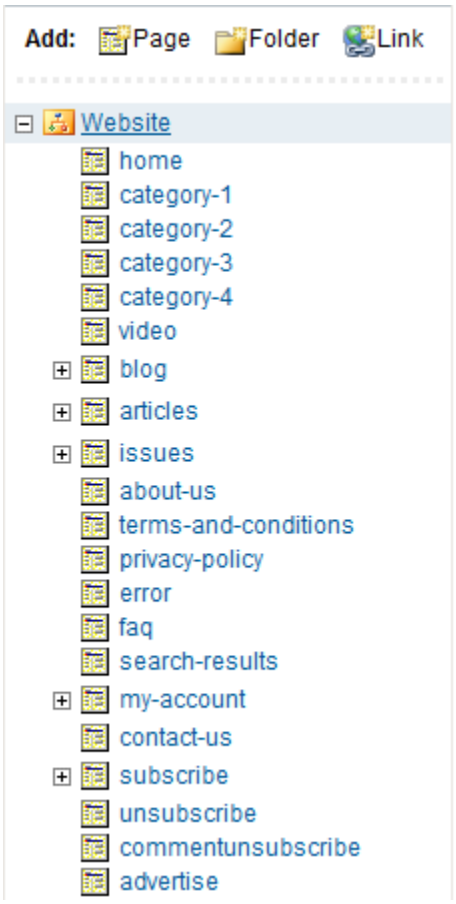
The Agility Digital Content Management component is used as a foundation for websites, mobile sites, and other "http delivered" content. In simple terms, it consists of lists of content items (such as blog post, news articles etc.) and pages (which in turn consist of templates and modules). The following diagram illustrates the relationship between these components:



This section of Developer 101 will break down and discuss each element of Web Content Management and walkthrough some labs to help demonstrate.

Pages

Pages in Agility are the main representation of a website's structure. You can see the existing structure of our Sample Site by clicking on the "Pages" link at the top of Agility.



In general, there are three simple types:

1. **Page** – this is an actual web page with a template and therefore module zones. There are a few basic properties
    - **Page Type** – this can be either Static or Dynamic Page. A static page is what we consider a 'normal' page; it has a fixed name and contains a number of modules with relatively simple content. A dynamic page implements URL routing and is linked to a Dynamic Page List. This will be discussed in more detail later on.

- **Menu Text** – this is the 'link' text that will be used when the page appears on a navigation control or sitemap
- **Page Title** – this is the text that appears in "title" element of the raw HTML, and is therefore used as the main title in the browser
- **Page Name** – this is the name of the page within the context of the website: for example, a page with the name "home" would be accessed via the url "~/home"
- **Page Template** – this is the actual web page that is used as basis for building up the page content. It contains modules zones that can have modules added to them. We'll get to the details on this later

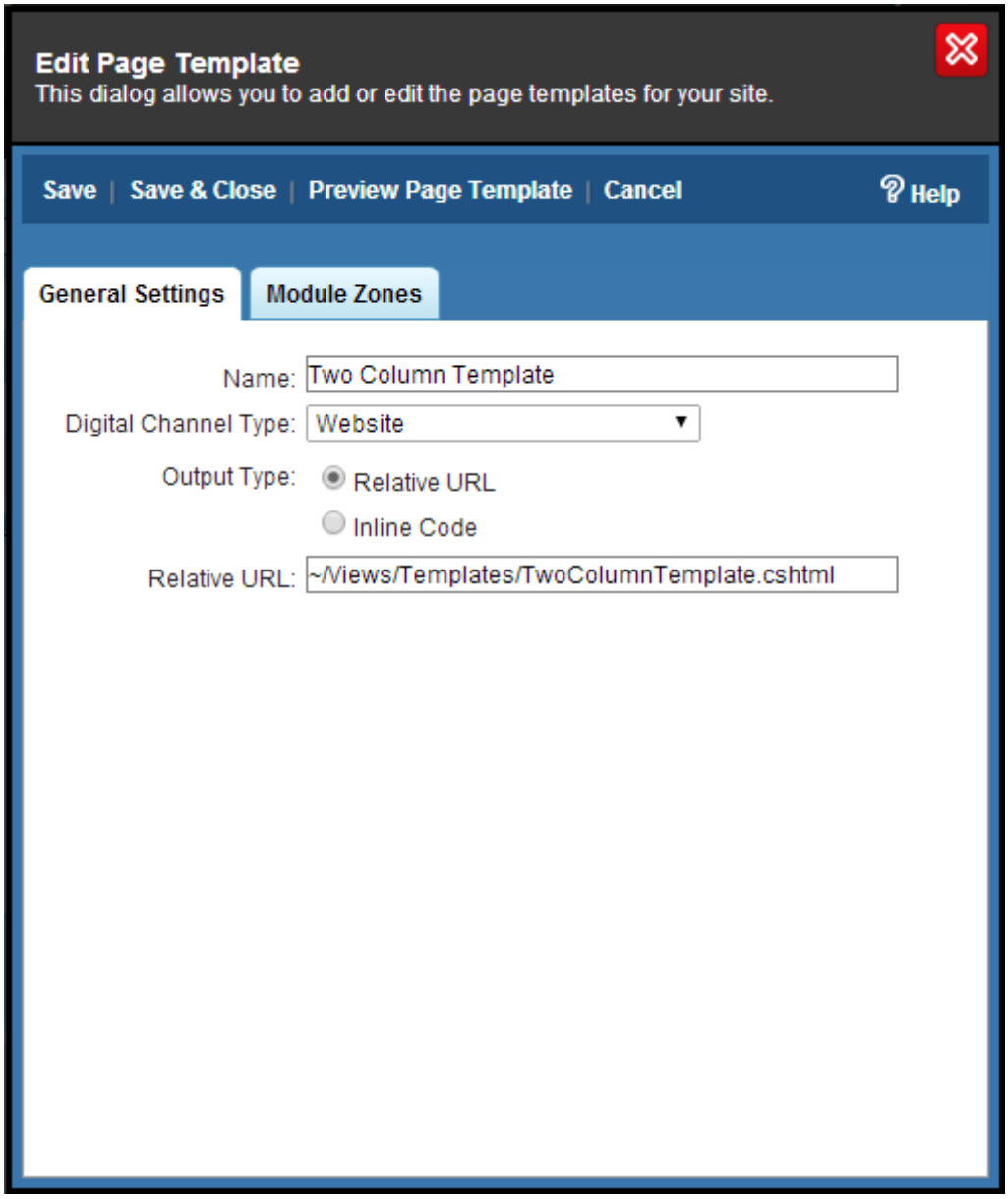
2. **Folder** – this is a virtual folder to group pages. It cannot be accessed as a resource in its own right. This also has a couple of basic properties:
    - **Menu Text** – the same as in a page, this is the text that appears as the directory on links in a navigation control or on a sitemap
    - **Folder Name** – similar to the page name above, this is the name that represents the directory in the context of a website: so a folder named "news" means that all pages below it would start with "~/news/"
3. **Link** – this can be used to place a link in the map to another page, a document, or to a completely different website
    - **Menu Text** – the same as in pages and folders, this is the text that appears as the directory on links in a navigation control or on a sitemap
    - **Link Name** – this will form the URL of the link in site map, relative to its position in the page tree
    - **Link** – this is the target resource to which the link will redirect. Agility allows you to pick another page within the page tree, a document within the document tree or a completely custom URL
    - **Target** – the target window to open the link into

Page Templates

In a typical web solution based on Agility, there will be a number of templates used to define the layout of a page. You can see how we have set up our Sample Site in Agility, by selecting the Settings link at the top of the page and then clicking on Page Templates. Here, you will see multiple page templates. This means that we have a single template used to define the layout of the home page, and various other templates used to define the layout of all the other pages in the website. You can have as many templates as you choose, but it is good practice to narrow down the designs as much as possible so that content editors have a concise list to choose from.

A page template consists of three basic properties in the General Settings tab (click on any of the page templates to open its properties dialog):

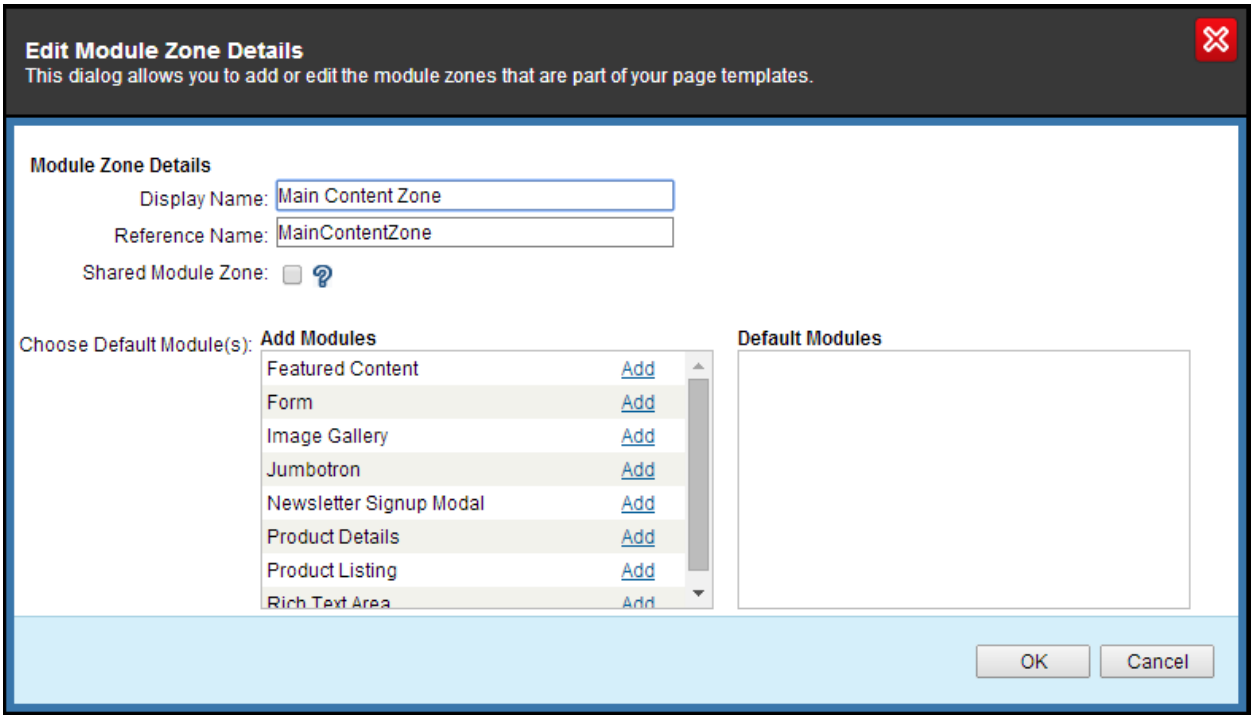
- **Name** – this is simply a title used to identify the template
- **Output Type** – You can choose where to store your Page Template options include a Relative URL or Inline Code
- **Relative URL** – this is a URL relative to your website path (if not storing in Inline Code) that locates the view (.cshtml file) of the template.
- **Digital Channel Type** – this defines which type of channel the template can be used on (such as Website or Mobile Site). For the purpose of this training, we will always use Website.



A page template then has a second tab containing a list of all the module zones - these are the areas in which a content editor can add modules when creating a page. In the actual .cshtml view, the module is called with Razor syntax and passes the module zone's reference name as a parameter. On a typical two column template, this might consist of a Main Content Zone and a Sidebar Content Zone. Each of these in turn has a number of properties:

- **Display Name** – similar to the Name of the title, this is simply the identifier of the zone that the content editor will see when they are creating a page.

- **Reference Name** – this is what you will use when defining the div elements on your physical template page. The reference name will be passed to the RenderContentZone() method, and used by Agility to work out where to render each module.
- **Shared Module Zone** – this optionally specifies the Content Zone as a shared set of modules. This means that any page that uses this Page Template will always have the exact same modules in this Content Zone. If you add, remove, or change a module within this Content Zone on any page, then that change will propagate to all other pages that share this Page Template. This is often used in cases where a section of a site will always share the same modules in a specified content zone and allows you to have a central point of updating these modules without having to change it manually on multiple pages.
- **Default modules** – this is often used by Architects to define a default structure of a set of pages. For example, you may have page template that defines a certain area of a website like News, and you may want to ensure a certain module appears on each of these pages (like "Top Stories" or "Most Read") – in this case, you would add this module to the default modules list here. Content editors can always remove the modules if they don't want them on a particular page.



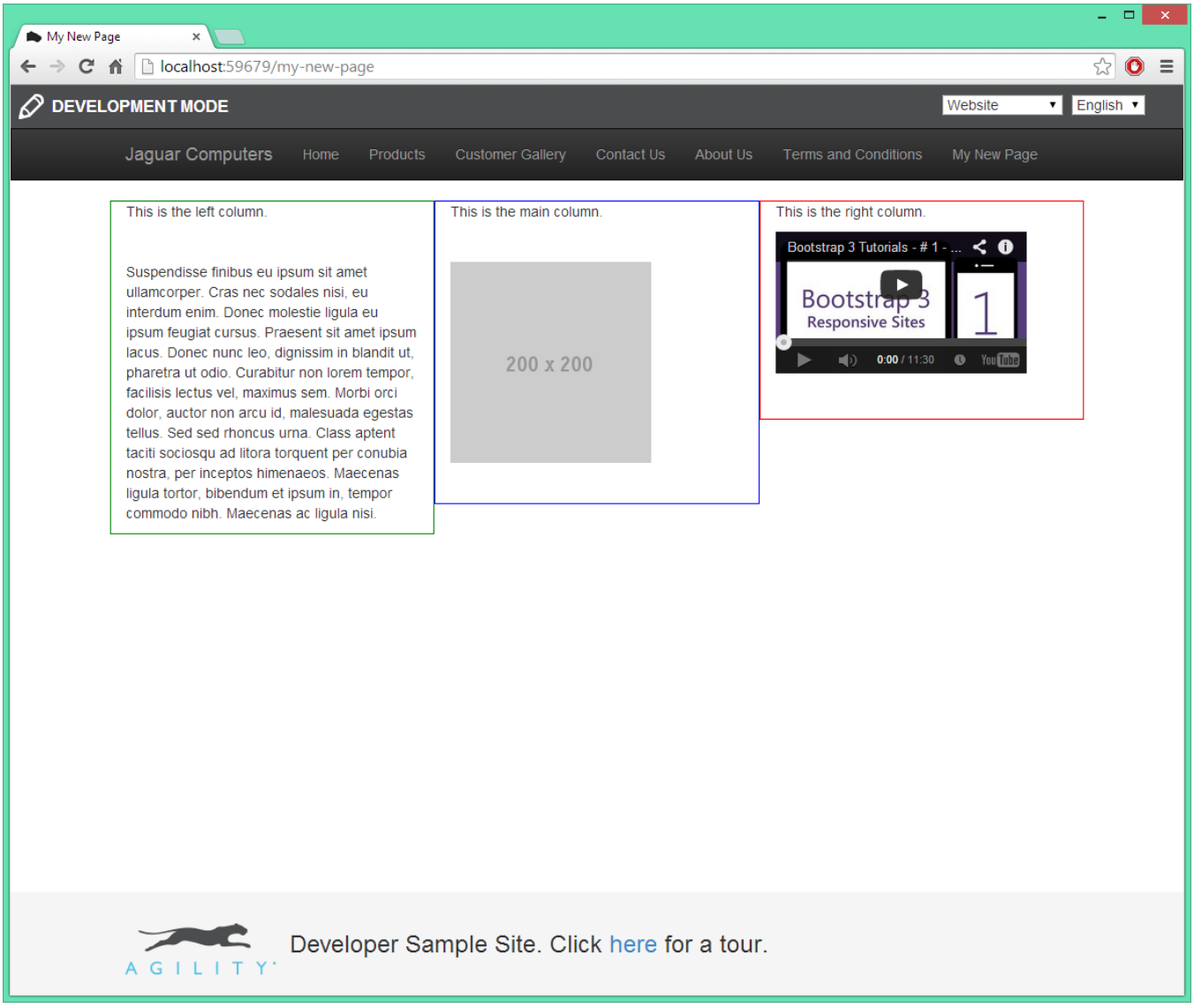
Lab – Define and Create a Page Template

In this lab, we will create a working page with a new Page Template. You will need to have the Agility Content Manager open in a web browser and our Sample Site open in Visual Studio. At the end of the lab, you will have a three column template with a page containing some rich text.

1. In the **Settings** section of Agility, select **Page Templates** and click on **New Page Template**.
2. Name the template "Three Column Template", select Relative URL as the Output Type and enter the Relative URL as "~/Views/Templates/ThreeColumnTemplate.cshtml". Click on **Save** to enable the **Module Zones** tab.
3. Now click on the **Module Zones** tab, and then **New Module Zone**.
4. Name the content zone "Left Content Zone" with a reference name of "LeftContentZone"
5. Repeat step 4 to create "Main Content Zone" and "Right Content Zone", with appropriate reference names.
6. Click on **OK** and then **Save & Close** to complete the creation of the page template.
7. Now we will need to create the physical file in the website. In Visual Studio, add a new partial view in the Views\Templates directory called "ThreeColumnTemplate.cshtml".
8. In the HTML source of the page, add three divs to act as containers for your module zones. Add some styles or classes so you identify them.
9. Within each div, call the **RenderContentZone** method, passing the appropriate reference name as a parameter:

```
<div class="container theme-showcase three-column-template contentWrap" role="main">
    <div class="row">

        <div class="col-md-4">
            @{Html.RenderContentZone("LeftContentZone");}
        </div>

        <div class="col-md-4">
            @{Html.RenderContentZone("MainContentZone");}
        </div>

        <div class="col-md-4">
            @{Html.RenderContentZone("RightContentZone");}
        </div>

    </div>
</div>
```

10. Back in **Agility**, navigate to **Pages**. Click on **Page** on the 'Add' submenu to create a new page. Enter the Menu Text as "My New Page", and the Page Title and Page Name will auto-populate
11. From the Page Template drop down list, select "Three Column Template", and click **OK** to save the new page.
12. You will see the content zones that we created in the right hand side of the page. Click on the **+Add** button to add a new module to each of the zones. Locate and select **Rich Text Area** in the list, and enter some HTML in the rich text editor that appears. Click on **Save & Close** to save this module.
13. Now, in a browser, navigate to the **http://localhost:{port-number}/my-new-page**, where you will see the HTML that you just entered into the rich text area modules.


My New Page
localhost:59679/my-new-page
DEVELOPMENT MODE
Website
English
Jaguar Computers
Home
Products
Customer Gallery
Contact Us
About Us
Terms and Conditions
My New Page
This is the left column.
Suspendisse finibus eu ipsum sit amet ullamcorper. Cras nec sodales nisi, eu interdum enim. Donec molestie ligula eu ipsum feugiat cursus. Praesent sit amet ipsum lacus. Donec nunc leo, dignissim in blandit ut, pharetra ut odio. Curabitur non lorem tempor, facilisis lectus vel, maximus sem. Morbi orci dolor, auctor non arcu id, malesuada egestas tellus. Sed sed rhoncus urna. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas ligula tortor, bibendum et ipsum in, tempor commodo nibh. Maecenas ac ligula nisi.
This is the main column.
200 x 200
This is the right column.
Bootstrap 3 Tutorials - # 1 - ...
Bootstrap 3
Responsive Sites
1
0:00 / 11:30
AGILITY
Developer Sample Site. Click here for a tour.

Lab – Define and Create a Page Template with a Shared Module Zone

In this lab, we will create two working pages with a new Page Template using a Shared Module Zone. Using Shared Module Zones, you can ensure that any page created using that same Page Template will all share the same Modules (and their data) across each page. Changing the modules in that zone will propagate to all other pages that share that same Page Template. Shared Module Zones are important to consider when architecting your pages as they can vastly improve your ability to manage your content. In this example, we will create a similar Three Column Template with an additional Feature Content Zone that will be a Shared Module Zone and span the width of the page.

1. In the Settings section of **Agility**, select **Page Templates** and click on **New Page Template**.
2. Name the template "Three Column Feature Template (Shared Feature)", select Relative URL as the Output Type and enter the Relative URL as "~/Views/Templates/ThreeColumnFeatureTemplate.cshtml". Click on **Save** to enable the **Module Zones** tab.
3. Now click on the **Module Zones** tab, and then **New Module Zone**.
4. Name the content zone "Feature Content Zone" with a reference name of "FeatureContentZone".
5. Check off the checkbox labelled **Shared Module Zone**, this will enable the Feature Content Zone to be a **Shared Module Zone.**
6. Repeat step 4 to create your three columns "Left Content Zone", "Main Content Zone" and "Right Content Zone", with appropriate reference names.
7. Click on **OK** and then **Save & Close** to complete the creation of the page template.
8. Now we will need to create the physical file in the website. In Visual Studio, add a new partial view in the Views\Templates directory called "ThreeColumnFeatureTemplate.cshtml".
9. In the HTML source of the page, add four divs to act as containers for your module zones. Add some styles or classes so you identify them.
10. Within each div, call the **RenderContentZone** method, passing the appropriate reference name as a parameter:

```
<div class="container theme-showcase three-column-template contentWrap" role="main">
    <div class="row">

        <div class="col-12-md">
            @{Html.RenderContentZone("FeatureContentZone");}
        </div>

    </div>

    <div class="row">

        <div class="col-md-4">
            @{Html.RenderContentZone("LeftContentZone");}
        </div>

        <div class="col-md-4">
            @{Html.RenderContentZone("MainContentZone");}
        </div>

        <div class="col-md-4">
            @{Html.RenderContentZone("RightContentZone");}
        </div>

    </div>
</div>
```

11. Back in **Agility**, navigate to **Pages**. Click on **Page** on the 'Add' submenu to create a new page. Enter the Menu Text as "Shared Feature Page 1", and the Page Title and Page Name will auto-populate
12. From the Page Template drop down list, select "Three Column Feature Template (Shared Feature)", and click **OK** to save the new page.
13. You will see the content zones that we created in the right hand side of the page. Click on the **+Add** button to add a new module to each of the zones. Locate and select **Rich Text Area** in the list, and enter some HTML in the rich text editor that appears. Click on **Save & Close** to save this module.
14. Repeat step 11-13, creating additional unique pages based on this Page Template and note the Shared Feature Content Zone.
15. Now, in a browser, navigate to your pages and note they all share the same Feature Content Zone while maintaining their own unique content in the Left, Main, and Right Content Zones.

Modules

In the "Define and create a Page Template" lab, we used the built-in Rich Text Module to illustrate how modules can be added to a page. Now we can discuss modules in more detail so that we can create a custom module to implement some specific functionality.

Modules are the individual functional components that populate a page layout. Since the module framework is basic and allows a high-degree of customization, the complexity of a module can vary from displaying a simple piece of text to a more complex form that collects and stores data somewhere.

From a developer's point-of-view, a module is either a Partial View or Controller ActionResult combination that encapsulates a specific section of the web page. This module then will include the mark-up to render the module, as well as any relevant business logic and JavaScript to make it interactive.

From the content editor's point-of-view, a module is an input form that appears when they select a module when adding it to a module zone on a page. This input form is the list of customizable properties of the module, such as titles, text labels, links, etc.

You can see the list of modules that already make up our Sample Site by navigating to **Settings** > **Module Definitions**. Some of these modules are of the more complex variety, and depend on data such as query strings and dynamic routing via Dynamic Pages. There are also more simple modules (such as the Badge) that simply render an image and link. You can click on any of the module definitions to see its properties dialog, summarized as follows:

1. General Settings
    - **Name** – this is simply the reference name displayed to content editors when they are searching for a module to add to a content zone, or reviewing the modules that have already been added.
    - **Reference Name** – similar to the reference name of a page Content Zone in a page template, this is a name for the website code to use when referring to the module.
    - **Description** – this is a mandatory field that is used to help content editors see what functionality they can expect from a module when adding it to a content zone.
    - The remaining checkboxes are important, but not relevant for this introduction to Agility
2. Properties
    - This tab is where you define the individual properties that make up the input form of a module – these are defined in the Architecture stage of website development and their usage can vary

- Properties can have various types, including Text, HTML, Number, Date, URL, Image, Image Gallery, File, and Linked Content. This wide variety in itself illustrates how diverse the usage of properties can be and will be discussed in the later labs.
- Tabs can be used to group properties together – all properties underneath a tab will appear grouped when the module's input form is rendered. For example, a module that collects data might group all of the field labels in one tab and all of the validation messages in another.
- Custom sections are used to insert rich text into a module input form – this can be for something simple like describing to a content editor how a module property is used, or even for inserting custom JavaScript to validate the data that a content editor enters.

3. **Form customization** – this is used to define how the content editor enters values for the various properties you define on your module. Typically, you will use the System Generated Input Form as this helps to maintain consistency for the content editors and makes maintenance of modules much easier. For a more complex module, you may wish to generate your own input form using the Input Form API.
4. **Custom Scripts** – this is used to define custom JavaScript code in your input form, which is executed as the form loads. Again, this is generally only used on more complex modules where heavy customization of the input form is required and you are interacting with the Input Form API
5. **Output template** – this defines what code will be executed when this module is being rendered in a page. For MVC projects, you can either specify either in Inline Code, a Controller Action, or a Partial View. Inline Code is a cloud alternative to storing code files within your website project, but is not relevant for this introduction. The Controller and Action option will simply execute the specified Action, passing through all of the module properties in a strongly typed object. The Partial View option will use the built-in Agility controller to execute the specified partial view, again making the module properties available in a strongly typed object.
6. **Page Usage** – this is a list of pages that have this particular module implemented on them. This is for information only.

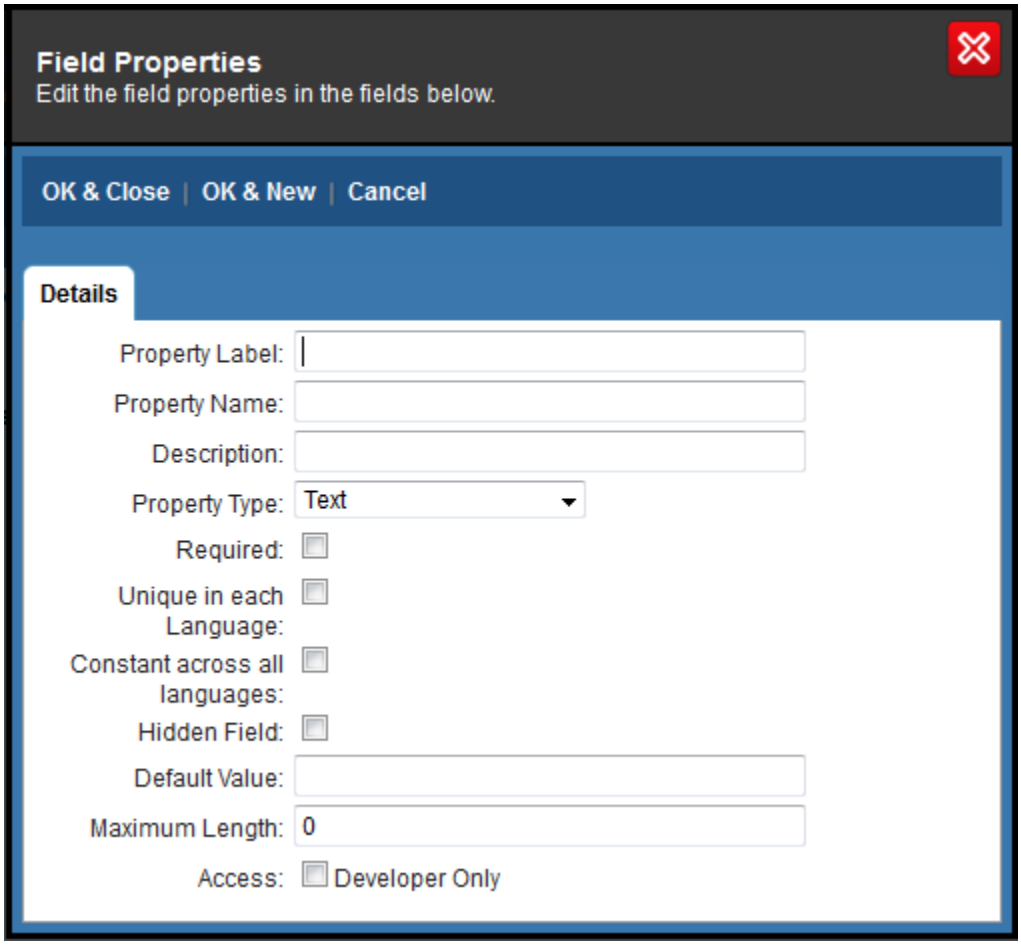
A Note on Modifying Module Definitions

Modifying a Module Definition will place the Module in a state where it needs to be published in order for the definition change to be synced to servers. If you attempt to publish a page which has an instance of that module on it, you will receive a sync error that will prevent the page from being published.

Lab – Create an Image Link Module

In this lab, we will create a module that displays an image wrapped in an anchor. Again, you will need to have the Agility Content Manager open in a web browser and our Sample Site open in Visual Studio. At the end of the lab, you will have created a module definition and added an instance of it to the page you created called "My New Page".

7. First, we need to define our module definition. Navigate to the **Settings** page in **Agility**, and select **Module Definitions**.
8. Click on **New Module Definition**. Give the new module a Name of "Image Link" - the Reference Name will be auto-populated. It is important to add a meaningful description here so that content editors know what to expect, give the value as "Displays a clickable image."
9. Click on **Save** to enable all the remaining tabs.
10. Now click on the **Properties** tab, so that we can start adding the properties of the module.
11. Click **Add Property** and the Properties dialog will appear. Give the first property a label of "Title", the Name field will auto-populate. Leave the property type drop-down as "Text" and leave the default value empty. Make sure the "Required" checkbox is checked, so that this property cannot by left blank by content editors.



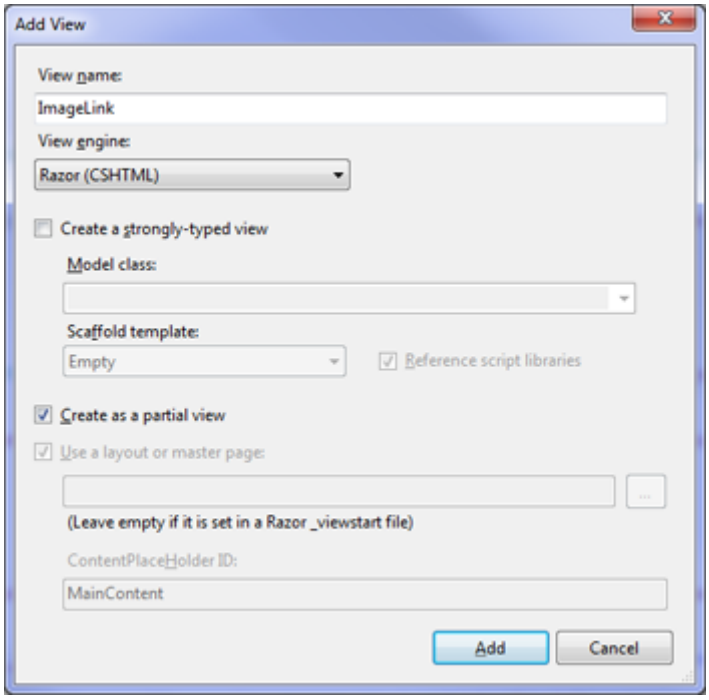
12. Click on **OK & New** to store this property and create another. Give the second property a label of "Image", and set the property type to "Image". You'll notice that the Image type enables some additional fields – File Size, Width, Height and Valid File Types. If your website design dictates that only a certain size of file will fit, you should populate these fields. For this example, we'll leave them as the defaults. You will also notice a tab called "Thumbnails" – this allows you to specify Thumbnail references so that you can automatically generate different sized images based on the Image being uploaded. This is important, but will not be covered in this introduction.

13. Click on **OK & New** again to store this property and create another new one. Give the third property and Label of "Link URL", and set the property type to "URL". Make sure the "Required" checkbox is checked, and click on **OK & Close**.

14. You now have the three properties that we need defined. Click on the **Output Template** tab, and select **Partial View on Website**. Set the Partial View Path to "~/Views/Modules/ImageLink.cshtml". Click on **Save & Close** to finish creating the module

definition. You may see a warning message at this point regarding publishing the content definition – this can be ignored for now.

15. Still in **Agility**, navigate to **Pages** and select the "my-new-page" page that you created previously. In the **Main Content Zone**, click on **+Add** and locate the newly-created "Image Link" module. This will open the module's input form.
16. Usually, you would take this opportunity to view the Module's input form from a content editor's perspective and fill in the form. Do the fields make sense? Is there enough instruction for the content editor to be able to fill out the form in confidence? If not, go back to the drawing board and re-adjust your module properties. If you are satisfied continue filling in the form.
17. We need to populate the Title, Image and Link URL fields. For the image, click on **Browse**, and then **Upload File** in the documents dialog that appears. Locate the file that you want to upload. Click on **Open** and the file will be uploaded - click on **OK** to attach the image to the module. The file is now selected, and you can add a meaningful value to the Alt Text field for display on the website. Note that the Alt Text field is not mandatory, but it is good practice to have one for SEO and Accessibility.
18. Next populate the Link URL by entering a URL of your choice. Note that you can also utilise the URL field by selecting a document from the Media & Documents list or a page from your sitemap by clicking the **Choose File** or **Choose Page** link. Enter the title of the link as appropriate, and change the target drop-down to "New Window".
19. Click **Save & Close** to finish editing this instance of the module.
20. We can now create the partial view that will display the module in the website code. Switch to Visual Studio and a new partial view in the Views/Modules directory called "ImageLink.cshtml".



21. Next, switch back to **Agility** and navigate to **Settings** > **Developer Downloads**. Click on the link **Content Definition API**, to download the latest Model objects for your project file. You can also download this file from within **Settings > Module Definitions** and **Settings > Content Definitions**. Locate the existing file in your solution - usually located in the **Models** directory. You can then copy the objects from the new file into the existing one. Be careful with this, as the C# Namespace generated in downloaded API may not be the same as what you have defined in your website code. In this particular case, the downloaded API has the namespace of "SampleMVC" and your project's Models has a namespace of "MVC4SampleSite.Models". To get around this, simply copy all the classes without the Namespace and replace the existing classes. Build your project to ensure there aren't any typos or missing braces.

22. We now need to tell the new View what type of data to expect. Assuming the namespace of your class is "MVC4SampleSite.Models" (check the top of your API file), add the following line to the top of your new partial view:

```
@model MVC4SampleSite.Models.Module_ImageLink
```

23.

24. We can now reference the Model's properties to build some markup for the view – taking full advantage of Visual Studio's intellisense. At this point you can create the markup however you wish, but use the example below to illustrate how to reference the Model's properties with Razor syntax:

```
@model MVC4SampleSite.Models.Module_ImageLink


<h1>@Model.Title</h1>
<a href="@Data.UrlEval(Model.LinkURL, "url")" target="@Data.UrlEval(Model.LinkURL,
"target")">
    <img src="@Model.Image.URL" alt="@Model.Image.Label" />
</a>
```

25. **Save** the file. In your browser, navigate to **http://localhost:{port-number}/my-new-page**. You will now be able to see your Image Link module displayed on the page.

Digital Content

Digital Content allows us to define lists or single items of data or information that do not fit into the model of pages and modules, but yet still need to be actively maintained by content editors. This could be a list of products, blog posts, or news articles, or even just a single item representing a header Partial View that is shared across multiple pages. These items can be global as 'Shared Content' items, or attached to a specific module as 'New Content'.

In many ways, these content items or lists are similar in concept to modules: we create a definition for each one and then create instances of it in order to add actual data items.

You can see the existing content definitions of our Sample Site by navigating to Settings > Content Definitions in Agility. You'll see that most of the items in the table have a type of "Content List", which makes perfect sense for list of products, categories, or featured items - where you'd expect there to be more than one item. You'll also notice that there's an item called "Global Header" which has a type of "Content Item" – in practice this also makes perfect sense as the website only has one header, we're using a content item here so that we can share it across every page on the website and only need to update its properties in one central place.

We don't need to go into exactly how lists and items are defined here, because they are constructed in exactly the same way as module definitions – you have the same property types available to you and the same input form customization options. The only addition to content lists is that you can specify what properties are displayed on the grid, and what order the items appear in.

A Note on Modifying Content Definitions

If you modify an existing content definition, that change will automatically be synchronized to your live web server. If you think this change will break something, it is good practice to **Deactivate** your live webserver until you have verified its working in your stage environment.

Linked Content Properties

Modules and Content Definitions can reference other Content Definitions and their Shared Content instances. This allows for shortcuts to other content, selecting a specific item from that linked list to use in its own context, and flexible content architecture.

Typically, the way you interact with your Linked Content is by selecting or sorting items from it through a variety of Rendered Controls such as a Drop Down List, Checkbox List, or a Grid. Selecting or sorting items will allow you to save the values of the Content IDs that represent each item. Using these saved ID(s) you can filter your Linked Content in C# using the Agility.Web API.

A Linked Content property within a Content Definition or Module Definition has its own set of settings:

- **Content Definition:** This is the list of Content Definitions in your Agility instance
- Content View:
    - **User Selectable:** This allows the content editor to select an item or list in Digital Content that shares this same Content Definition.
    - **Shared Content**: This allows the developer to specify exactly which item or list in Digital Content that should be used.
    - **New Content:** This creates a new instance of the item or list whenever an instance of this Module or Content Definition is saved.
- Render as:
    - **Grid:** Renders an embedded view of the selected content items
    - **Link:** Renders a link that will open a new dialogue displaying the details of this content item or items
    - **Dropdown List:** Renders a dropdown list of the selected content item or content items
    - **Checkbox List:** Renders a checkbox list of selected content item or content items
    - **Search List Box:** Renders a search box list that allows the content editor to search the selected content list and select one or many

When you are coding your Module, you can get reference to the Linked Content property by Name. For example, if the Name of your Linked Content property "Products" then you can get a reference to it in C# by using module. Products just like another other module property. Your Linked Content property uses the class IAgilityContentRepository<T> where T is the type of property (Content Definition) from your C# strongly typed API downloaded from Agility. For example, our module.Products would be IAgilityContentRepository<Product>.

Within the IAgilityContentRepostory class, we can filter the List or Item via the following calls:

```
//filter products by a particular category
    int categoryID = 20; //ContentID of the category
    string categoryRowFilter = string.Format("ProductCategoryID = {0}", categoryID);
    string sortByTitle = "Title ASC";
    IList<Product> productsFilteredByCategory = module.Products.Items(categoryRowFilter,
sortByTitle);


//get a subset of products from the list
    int skip = 10; //number of products to skip
    int take = 10; //number of products to take
    string rowFilter = ""; //not filtering by anything
    IList<Product> products = module.Products.Items(rowFilter, sortByTitle, take, skip);


//get products by their exact Content IDs
    int contentID = 2343;
    string contentIDs = "1232,43543,234";

    //GetById and GetItemsByIDs is found in Agility.Web.Extentions
    Product productByID = module.Products.GetById(contentID);
    IList<Product> productsByIDs = module.Products.GetItemsByIDs(contentIDs);


//sort products by IDs
    string sortIDs = "23,19,20,22,21";
    IList<Product> productsSortedByIDs = module.Products.SortByIDs(sortIDs);
```

Getting Content without Linked Content Properties

There may be cases where your Module doesn't have a linked content property, but you still want to be able get your "Products". Or maybe you are getting a Shared Content Item like a Global Header and want to use that to output content in your layout. You can still get reference to their IAgilityContentRepository programmatically assuming you know their Content Reference Name.

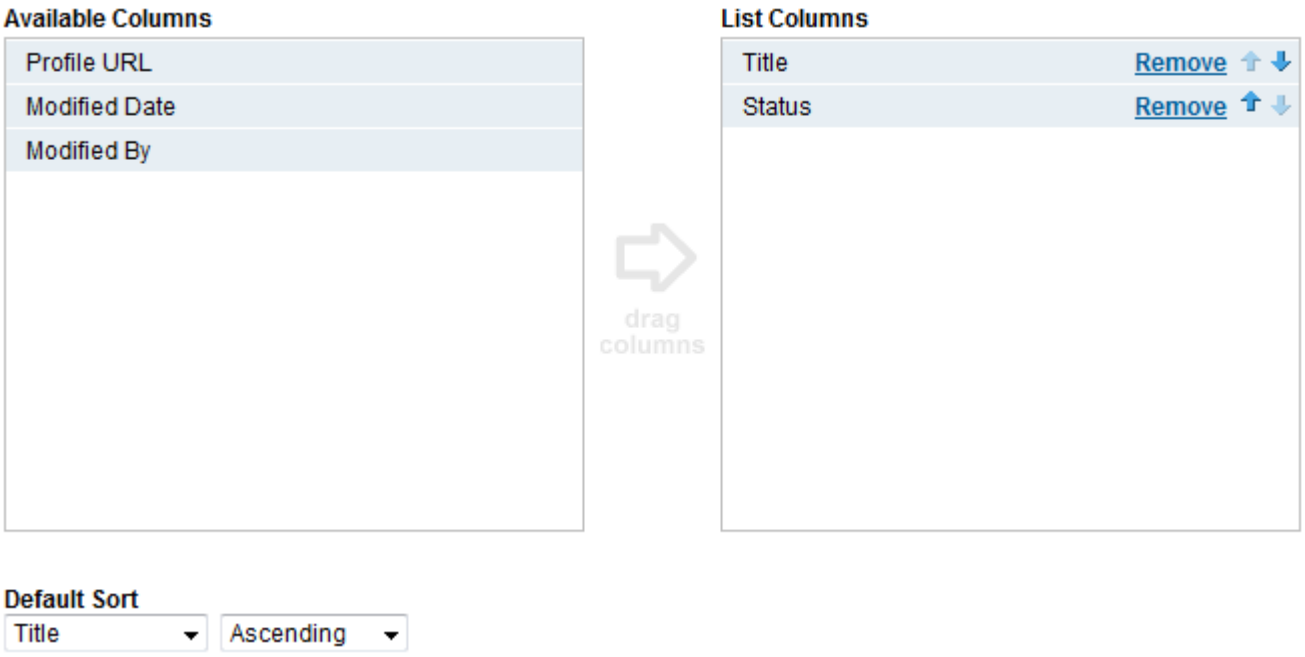
```
IAgilityContentRepository<Product> productsRepo = new
AgilityContentRepository<Product>("Products");
```

In this example "Products" is the Content Reference Name of the Shared Content List. Once you have the IAgilityContentRepository reference, you can then use the standard .Items(), .Item(), GetById(), GetItemsByIDs(), and SortByIDs() calls to filter or sort your content.

Lab – Create a list of Social Media Links

In this lab, we will define a list of social media accounts, and then create a module to display them on the website. Again, you will need to have the Agility Content Manager open in a web browser and our Sample Site open in Visual Studio. At the end of the lab, you will have a list of social media accounts on the page that you created "my-new-page".

26. Start by navigating to the **Settings** > **Content Definitions** in **Agility**.
27. Click on **New Content Definition** and set the Type to "Content List".
28. Enter the Name as "Social Media Accounts", and the Reference Name will auto-populate.
29. **Save** the content definition, and navigate to the Form Builder tab.
30. Click on **Add Field**, and enter the Field Label as "Title". We will be using this field as a way of ordering the data, later. Leave the field type as Text, ensure "Required" is checked, and click on **OK & New**.
31. Give the new property a label of "Profile URL", and set the field type to "URL". Ensure that "Required" is checked, set the Default Target to "New Window", and click on **OK & Close**.
32. Click on the **Default List Settings** tab. We'll define which properties we want to see on the data grids, and what order to display the items in. Drag the **Title** and **Status** fields from the **Available Columns** list into the **List Columns list**, and set the Default Sort to "Title".

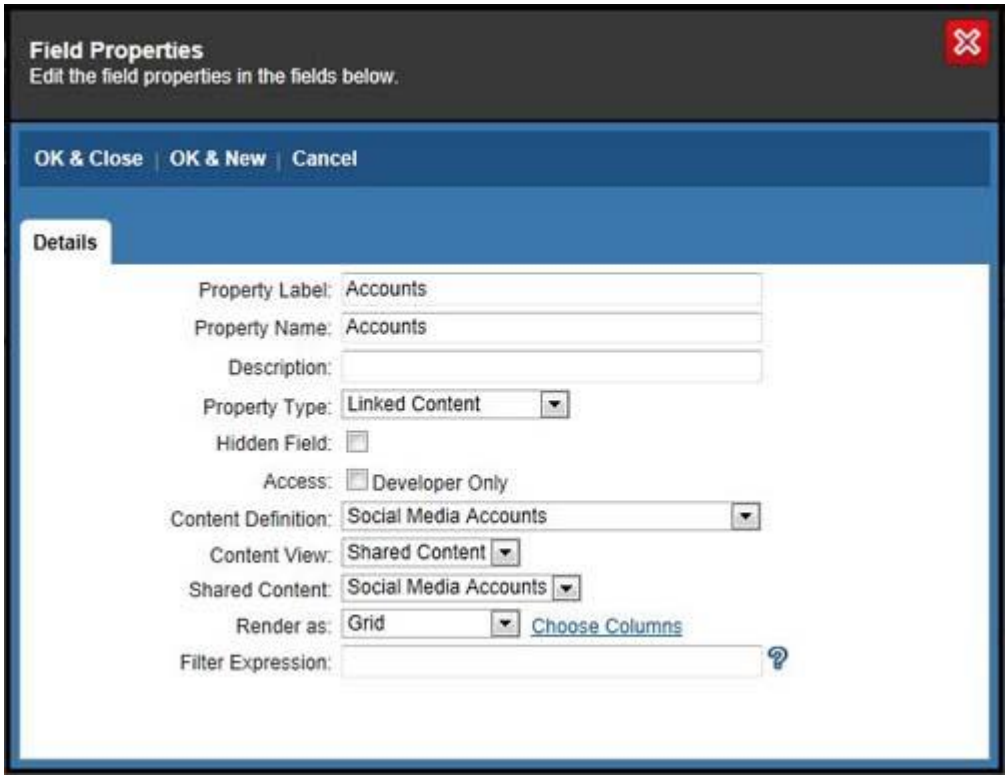


33. Now click on **Save & Close** to finish creating the content definition, and click on **OK** at the bottom of the screen to return to the Settings page.

34. We now need to create an 'instance' of the content definition so we can add data to it. Click on the **Digital Content** link at the top of the page, and then **New Content List**. In the dialog that opens, select "Social Media Accounts" from the Content Definition drop down list. Enter "Social Media Accounts" again as the Display Name (the Reference Name will auto-populate). Click **Save & Close** to finish create the list. You can now click on **Social Media Accounts** from the Shared Content list to open it, and add a number of test data items.

35. Now we have a list with data, we can create a module to display it on the website. In **Agility**, go back to **Settings > Module Definitions**. Create a new module called "Social Media Accounts Listing" - make sure the Reference Name and Description fields are populated appropriately.

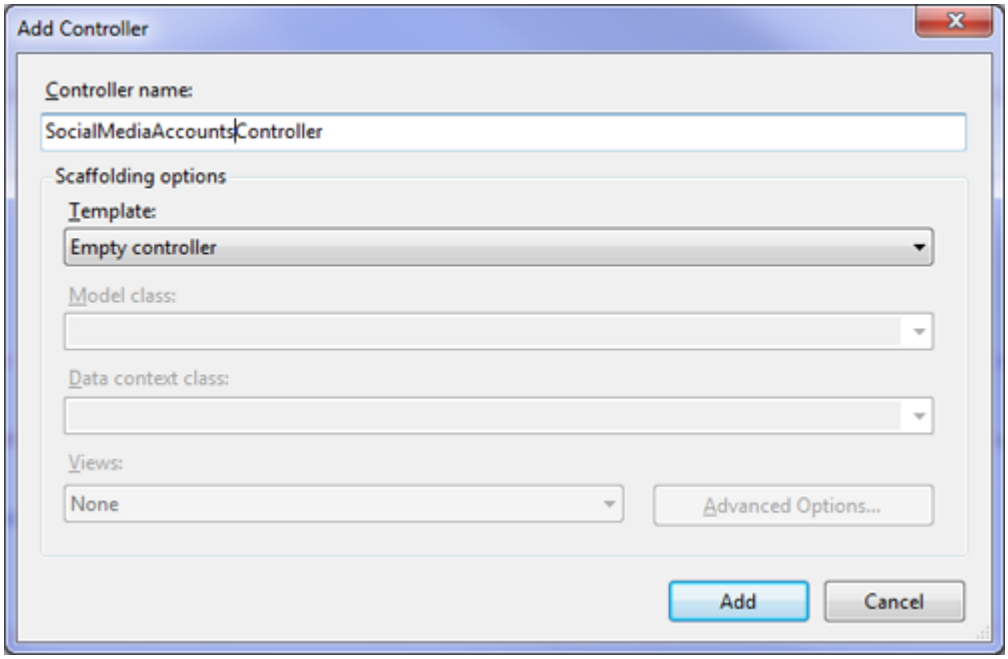
36. Select the **Properties** tab and click on **Add Property**. Enter the Label as "Accounts", change the property type to "Linked Content", the Content Definition to "Social Media Accounts" and the Content View to "Shared Content". Finally, select "Social Media Accounts" from the Shared Content list. This means that each instance of this module will reference a shared list of social media accounts. We can leave the "Render as" option as "Grid" to use the default settings we defined in the content definition.



37. Click **OK & Close** and then select the **Output Template** tab. Select the radio button option "MVC Controller Action", enter the "SocialMediaAccounts" as the Controller and "Listing" as the action. Click **Save & Close** to finish creating the module.

38. In the **Module Definitions** screen, click the **Download API** link to get the latest Model objects. Use this file download to update the contents of your API file, as we did in the previous lab.

39. We can now create the code files to go with the module. In the Controllers directory of your solution, add a new controller called "SocialMediaAccounts".



40. We now need to add an action to this controller for our new Listing module, with a parameter for the module's data. Your code should look like this:

```
using MVC4SampleSite.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVC4SampleSite.Controllers
{
    public class SocialMediaAccountsController : Controller
    {


        public ActionResult Listing(Module_SocialMediaAccounts module)
        {
            return PartialView(module);
        }

    }
}
```

41. Next, create the View to match - add a partial view called "Listing" in the directory Views/SocialMediaAccounts. We can now use Razor syntax again to output the module's properties (in this case, the linked list of social media accounts):

```
@using MVC4SampleSite.Models;
@model Module_SocialMediaAccounts

<ul>
    @foreach (SocialMediaAccounts account in Model.Accounts.Items())
    {
        <li>
            @account.Title: @Html.Raw(account.URL)
        </li>
    }
</ul>
```

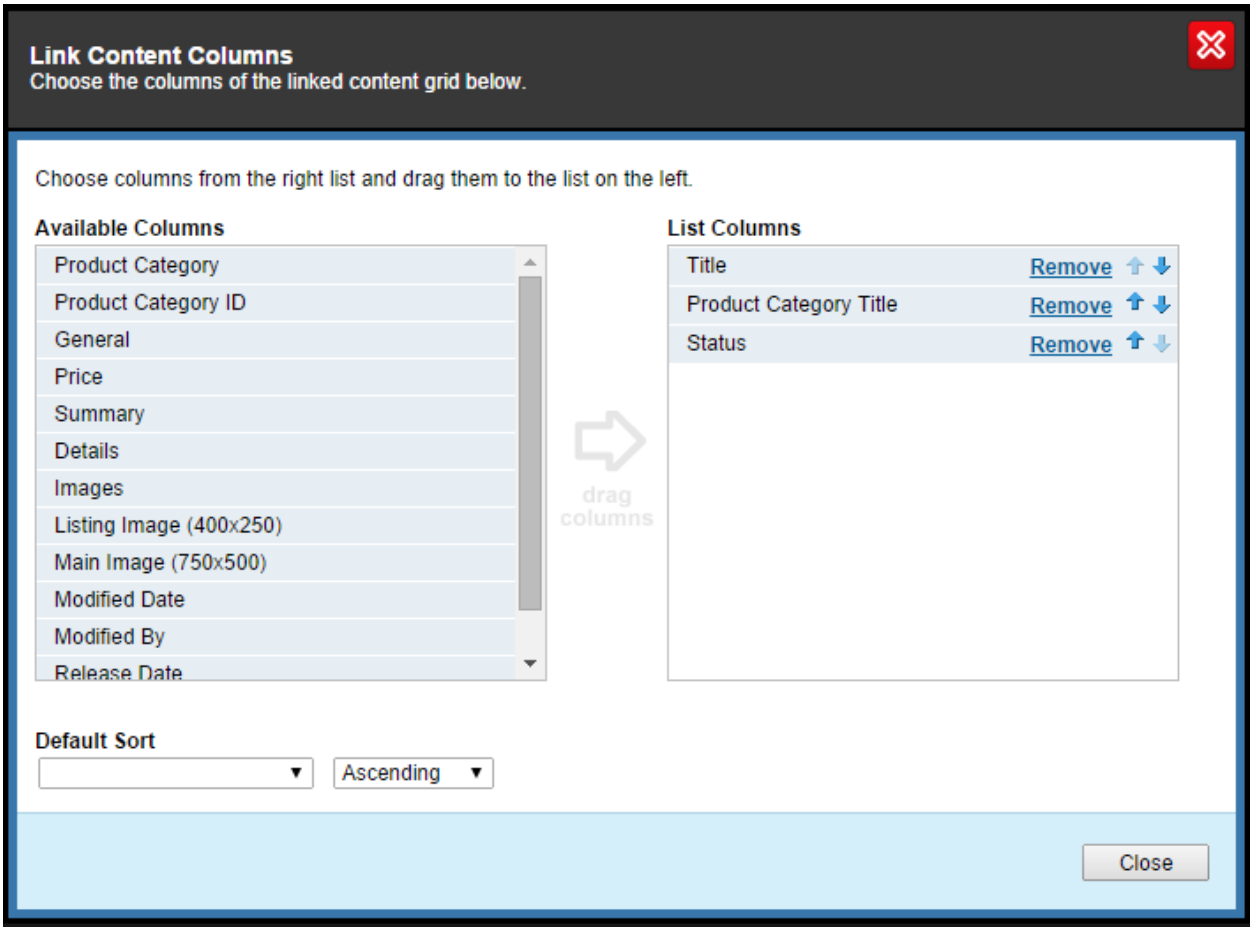
42. After saving these and files and building the solution, go back to **Agility** and add an instance of the new module on to the page we created “my-new-page”. After saving the page configuration, navigate to this page in your browser to see the module displayed.

Lab – Create a Featured Products Module

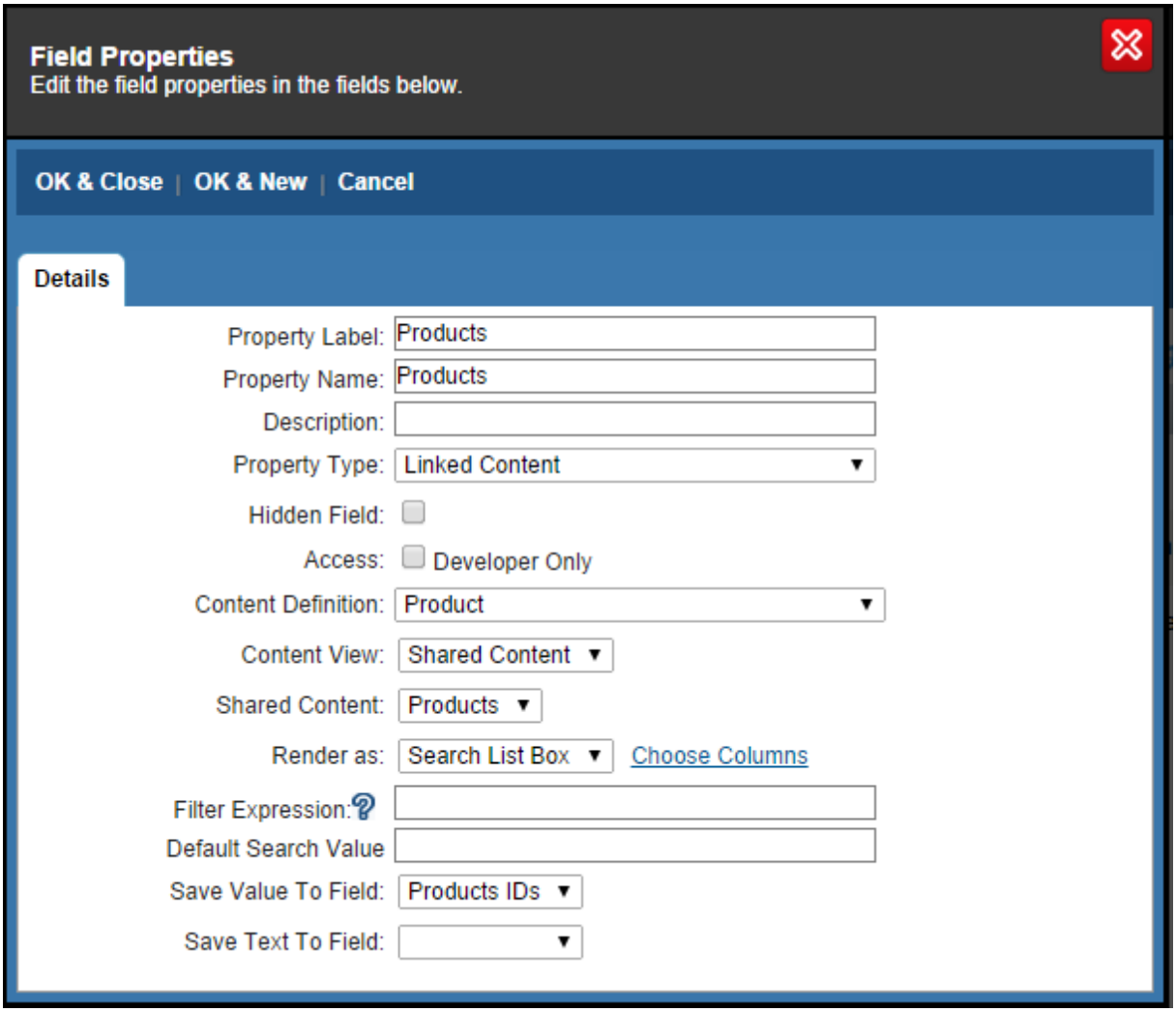
In this lab, you will be creating a Module that allows content editors to search and select from a list of available of "Products" and have them featured in a listing on the website. We will be making use of the existing Digital Content List found in **Digital Content > Dynamic Page Content** called "Products". Dynamic Page Content is a special type of Digital Content that allows content lists to be used dynamically as content for a Page. Each Product has its own dynamic page and is important, but not relevant to this exercise.

Again, you will need to have the Agility Content Manager open in a web browser and our Sample Site open in Visual Studio. At the end of the lab, you will have a list of featured "Products" and add this to the existing "home" page.

1. Start by navigating to **Settings > Module Definitions** in Agility.
2. Click New Module Definition.
3. Enter "Featured Products" as the Name and ensure the Reference Name is auto-populated.
4. Enter "Displays a list of Featured Products." as the Description.
5. Click **Save** to save this module.
6. This module will have several properties including "Title", "ProductsIDs", and "Products". Click on the **Properties** tab and click **Add Property**.
7. Enter "Title" as the Property Label and ensure the Property Name is also filled out. Set the Property Type to "Text". Click **OK & New**.
8. Enter "Products IDs" as the Property Label and ensure the Property Name is also filled out. Set the Property Type to "Text" and mark the field as "Hidden Value" – this will hide the text field from the input form. We'll need this later to store our selected and sorted Product IDs. Click **OK & New**.
9. Enter "Products" as the Property Label and ensure the Property Name is also filled out. Set the Property Type to "Linked Content". Additional fields relating to Linked Content will be shown.
10. Set the Content Definition to "Product".
11. Set the Content View to "Shared Content". This will show an additional field called Shared Content allowing you to select a Shared Content List that exists based on the "Product" content definition. Select "Products".
12. Set Render as "Search List Box". Additional fields relating to the Search List Box will be displayed.
13. Click the **Choose Columns** link and drag over "Title" and "Product Category Title" as well as the "Status" from the Available Columns to the List Columns.

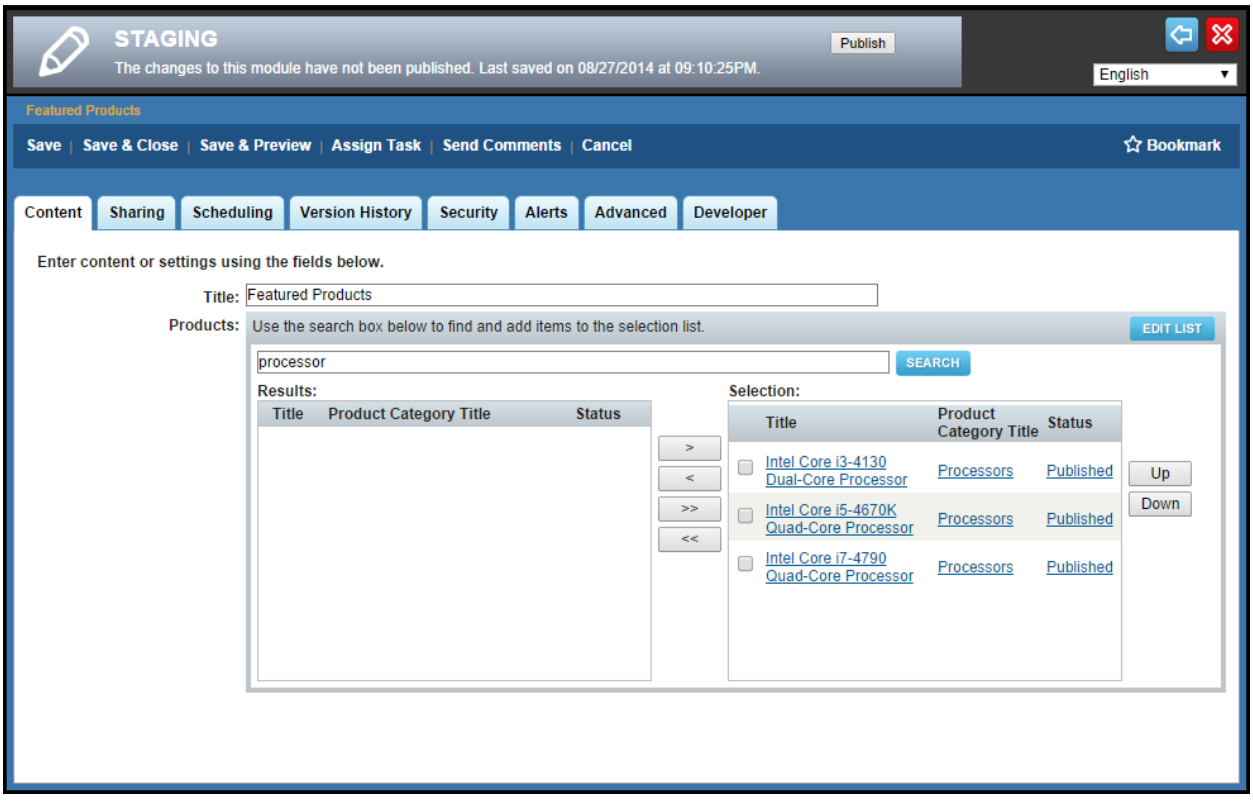


14. Leave the Filter Expression field blank.
15. Leave the Default Search Value field blank.
16. Set the Save Value To Field to your hidden property you created called "Products IDs".
17. The Save Text To Field property can be left blank in this case, although can be useful when wanting to store the Title of the Product in addition to its ID.



18. Click OK & Close.
19. Click on the **Output Template** tab.
20. Select "MVC Controller Action" as the Output Type and enter "Products" as the Controller and "Featured" as the Action.
21. Click **Save & Close** on the Edit Module Definition dialogue.
22. Click **Download API** to download your latest C# classes and replace the contents of the existing API file Models/MVC4SampleSite_API.cs.

23. In **Agility** navigate to **Pages** and click on the page labelled "home". Add the "Featured Products" module to the Main Content Zone.
24. Set the Title to "Featured Products".
25. Click **Save** to save the module and initialize the Linked Content property.
26. Enter "processor" into the search field and hit **Enter** on your keyboard or click **SEARCH** to search the Products list for any Product Title that contains "processor".
27. If you don't receive any results, change the search term and try again. You can always click on the **EDIT LIST** button to see the entire Products list.
28. Once you have some search results, click the checkbox beside the corresponding product(s) you like to select and click the **>** button to complete your selection. Note you can also select items from your Selection list and move them up or down to customize your sort order. Adding, removing, or sorting items will change the comma delimited list of IDs that are being stored in your hidden "Products IDs" field.



29. Once you are satisfied with your selection click **Save & Close** to close the module.

30. Next, we need to add in the appropriate Controller Action method and create our partial view for our module.

31. In **Visual Studio** open the Controller called "ProductsController". You'll notice we already have other Controller Action Methods here for our general Product Listing and Product Details modules. Add a new ActionResult near the bottom called "Featured" and we will pass in the Module_FeaturedProducts class as a parameter called "module".

```
public ActionResult Featured(Module_FeaturedProducts module)
{
     return PartialView("~/Views/Products/FeaturedProducts.cshtml", module);
}
```

32. Before we move onto creating our PartialView called "FeaturedProducts.chsml", let's add some validation to this module. We only want to output this module if we have any products that were selected. If the editor didn't select any products, or the products they selected have been unpublished or deleted then we don't want to display something that looks like empty content on the website. To get around this, we can add some logic that only renders the view appropriately.

```
public ActionResult Featured(Module_FeaturedProducts module)
{
    //check if the Products linked content is initialized and the 'Products' list still ex-
ists,
    //then get products by our selected IDs and see if we have any
    //if we do, then display then render the view, else return nothing
    if (module.Products != null && module.Products.GetItemsByIDs(module.ProductsIDs).Any())
    {
        return PartialView("~/Views/Products/FeaturedProducts.cshtml", module);
    }

    return null;

}
```

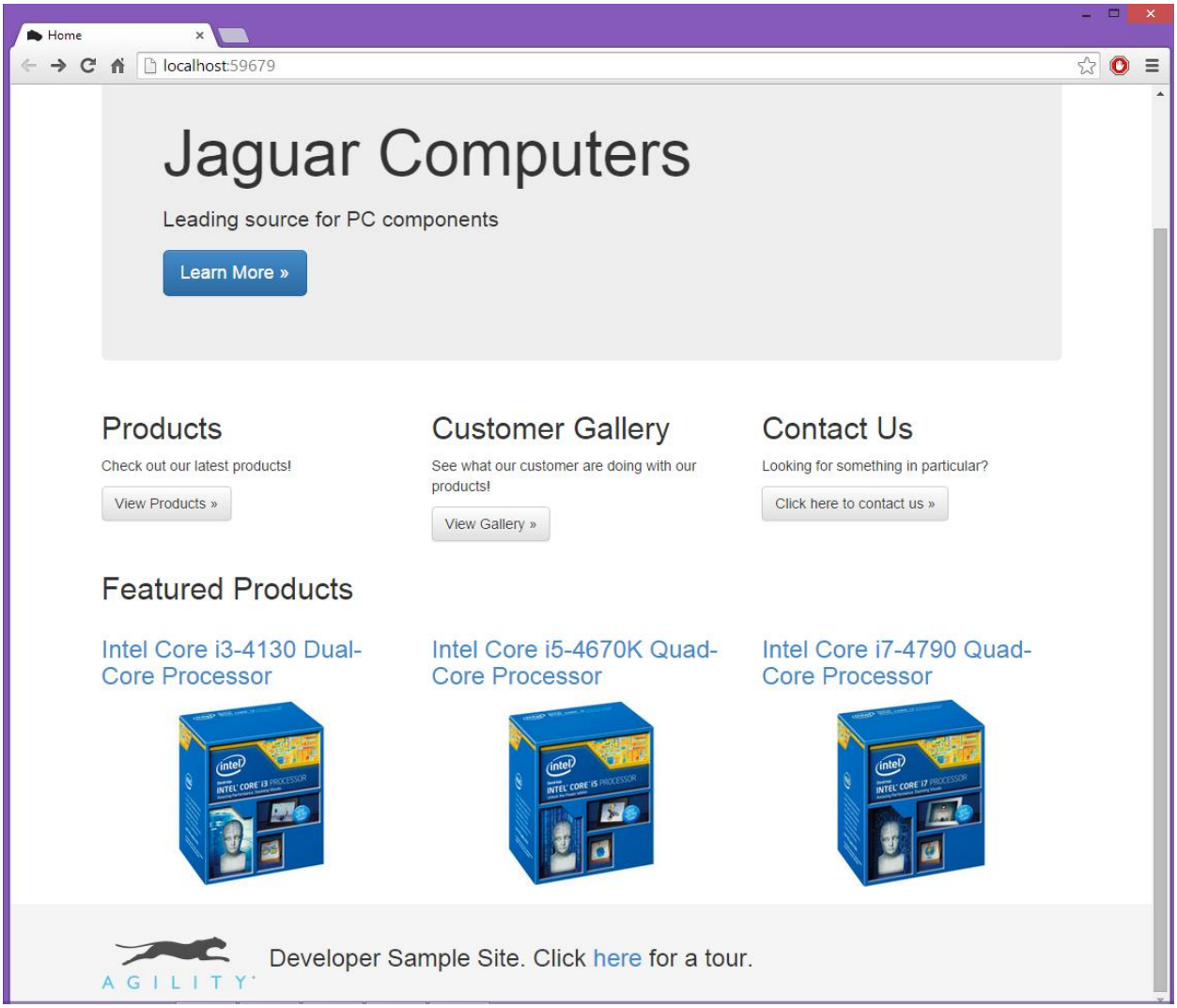
33. Create a new Partial View in Views/Products/ called "FeaturedProducts" and set the model of the view to Module_FeaturedProducts.

```
@model MVC4SampleSite.Models.Module_FeaturedProducts
```

34. Type in the following code into your view:

```
@* Adding Agility.Web.Extensions allows us to use the method GetItemsByIDs which filters
the content view by our IDs *@
@using Agility.Web.Extensions
@model MVC4SampleSite.Models.Module_FeaturedProducts

<div class="featured-products">

    @* If the Title is empty, don't display anything *@
    @if(!string.IsNullOrEmpty(Model.Title)) {
    <h2>@Model.Title</h2>
    }

    <div class="row">

        @foreach(var product in Model.Products.GetItemsByIDs(Model.ProductsIDs))
        {

        <div class="col-md-4">
            <a href="@product.Url("~/products/product-details")">
                <h3>@product.Title</h3>
            </a>
            @if(product.ListingImage != null) {
            <img src="@product.ListingImage.URL" class="img-responsive" />
            }
        </div>

        }

    </div>

</div>
```

35. Build your project and Start Without Debugging in **Visual Studio**. On the homepage, you should see your Featured Products module along with any products you have selected.


Home
localhost:59679
Jaguar Computers
Leading source for PC components
Learn More »
Products
Check out our latest products!
View Products »
Customer Gallery
See what our customer are doing with our products!
View Gallery »
Contact Us
Looking for something in particular?
Click here to contact us »
Featured Products
Intel Core i3-4130 Dual-Core Processor
Intel Core i5-4670K Quad-Core Processor
Intel Core i7-4790 Quad-Core Processor
AGILITY
Developer Sample Site. Click here for a tour.

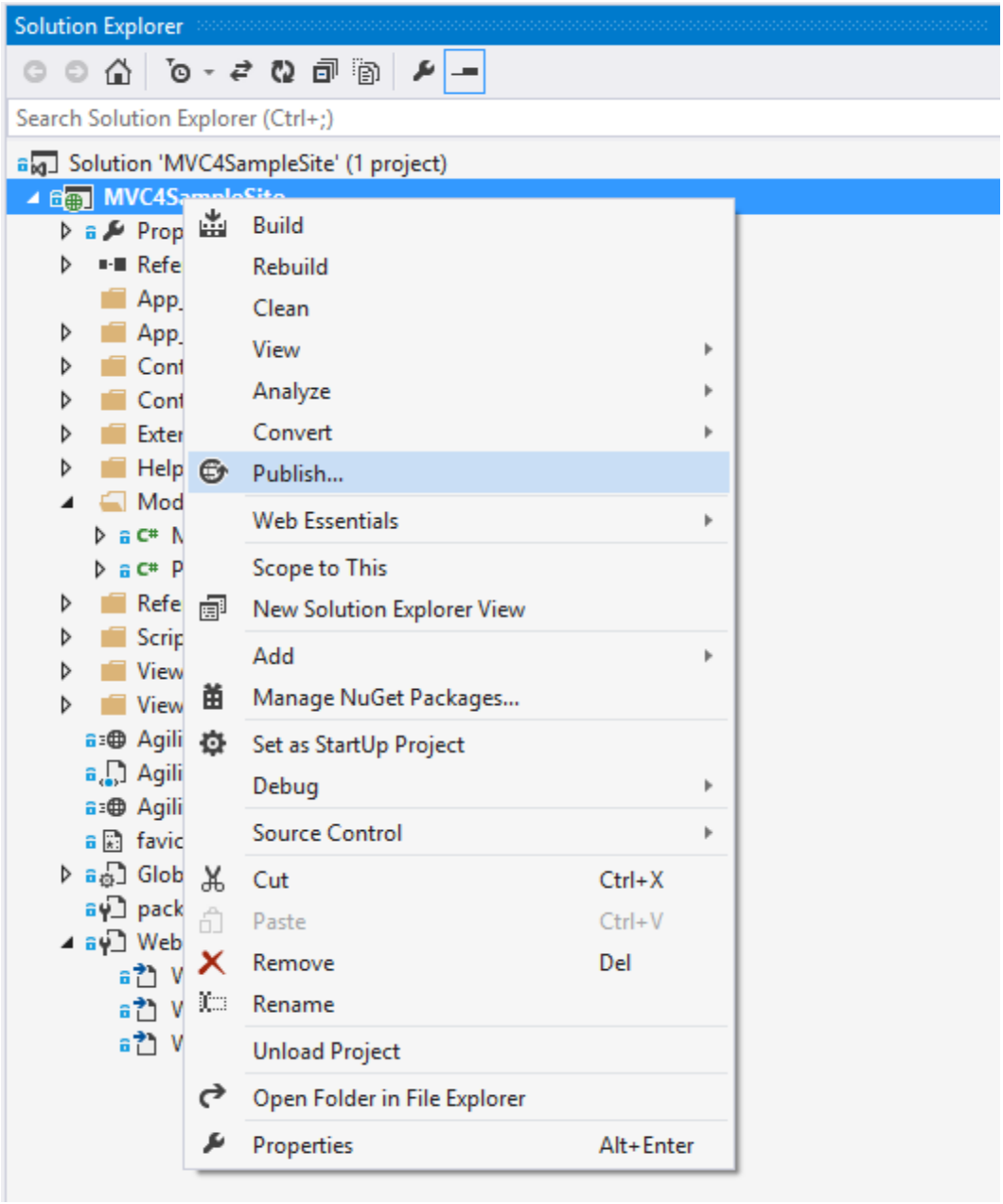
Lab – Website Deployments

Now that we've built some modules in the previous Labs and have verified they are working locally in Development Mode, it's time to deploy these to our server.

In this particular case, we only have one server we can deploy to – normally we would have a Stage and Live server for real websites. We will be using FTP to deploy your code changes. You will be provided the FTP credentials for your sample site.

Deploying our website requires us to publish our local project in Visual Studio – using this build, we can kick off a deployment which will replace the build we have on our server.

1. With your "MVC4SampleSite" project open in Visual Studio, ensure you are viewing the Solution Explorer Window and right-click the **MVC4SampleSite** project in the explorer tree. Click **Publish**.



2. You should see the "Publish Web" dialogue window open. This window allows you to configure how you want to publish your website. Publishing methods supported include FTP and File System. You should notice there are already two profiles created for you called "Staging" and "Release". Each of these profiles are set to publish the compiled code to their respective folder located in "C:\Builds\SampleMVC\". Each of these profile are also set to "Delete all existing files

prior to publish" – this ensures that any existing files in the directory are removed before publishing.

3. As you may have guessed, each publishing profile corresponds to a build configuration. A build configuration allows you to specify a Config Transform file that will be used in tandem with the Web.Config file to provide you a unique transformed Web.Config file based on your build configuration.

4. A common use case for this is managing how our Agility site functions based on different build configurations. For example, in order for our site to run locally it needs to be in Development Mode – this is controlled via a setting in the <agility.web> section of your Web.Config. However, we don't want to have to remember to change the setting each time you make a deployment to a server (accident waiting to happen), this is where the Staging and Release build configurations come into play.

5. Using the Staging or the Release build configuration, their respective transform files (Web.Staging.Config or Web.Release.Config) is applied to the standard web.config file – overwriting values specific to that build configuration such as `developmentMode="false"`.

6. To learn more about Config Transforms see: http://msdn.microsoft.com/en-us/library/dd465326(v=vs.110).aspx.

7. Select the "Staging" Profile and click **Publish** to publish your MVC project to your local folder "C:\Builds\SampleMVC\Stage".

8. Navigate to "C:\Builds\SampleMVC\Stage" to view your compiled code. Open the web.config and examine the content, notice `applicationName="Agility Sample MVC4 (Stage)"` found in the <agility.web> section of the web.config. This confirms we have published the staging build configuration.

9. Once you are satisfied with your code, open your favourite FTP client (we like FileZilla) and connect to the FTP host. Overwrite all existing files in the FTP directory with your local build. If you are deploying to ftp://hosting-ftp.agilitycms.com it will take 30-60 seconds for the deployment to reach your web server.

10. Verify your changes.

Assignment – Create a Testimonials Module

This assignment is designed for you to put Web Content Management concepts to the test. You will be completing this within the Sample Site we've been working in.

Here's what our "client" is requesting:

"*We need a way to highlight customer testimonials. These testimonials are not related to products, but rather our store in general. We already have a list of testimonials in a Word Document ready to go, but we need an easy way to manage this list on the website. We would like the ability to select and add one or many testimonials to any page. Each testimonial will have a Name, Image (optional), and the Testimonial comment itself.*" – Jaguar Computers

Hint

From these requirements we can figure out what we need to build here:

- Should be a Module they can place on any page
- Within the Module, they should be able to select which testimonial(s) appear
- The Module will have a linked content property to a Shared Content List of Testimonials
- A Testimonial has a Name, Image, and Comment

When you have completed your module, add it to a page for testing and deploy your code. Looking for feedback on your Testimonials module? Email support@agilitycms.com and we'd be happy to review with you.

Looking Ahead

Throughout this **Developer 101 Training**, you have learned the basics of developing a website using Agility CMS. We covered how Agility connects to your website, the sync process, creating pages, creating page templates, creating modules, retrieving digital content, working with linked content, and deploying your website code. These are the fundamentals of building an Agility CMS powered website, however Agility CMS has far more features and tools available to assist developer and content editors.

Ready for the next course? Here's a taste of what we'll be covering next in the **Developer Intermediate Training**:

- **Introduction to User Generated Content (UGC)** – learn how to define, save, and search content submitted through the website such as Contest Submissions.
- **Dynamic Page Content** – learn how to build pages based on Shared Content lists such as Blogs
- **Routing** – learn how to capture URL patterns and programmatically render Agility pages, allowing you manipulate the logic of your modules based on the URL
- **Thumbnailing** – learn how to create, and access auto-generated thumbnails from Image properties
- **Image Galleries –** learn how to use the Image Gallery property and use it to manage a collection of images from Media & Documents
- **Inline Code** – learn how to manage CSS, JS, Page Templates, and .csthml partial views directly in the browser
- **URL Redirections** – learn how to add and manage URL Redirection directly in the Content Manager
- **Multi-Language** – learn the dos and don'ts of developing multi-language sites
- **Page Meta Data, Styles and Scripts** – learn usage, and how they are outputted in your layout
- **Web Servers** – learn how to change the states of your web servers