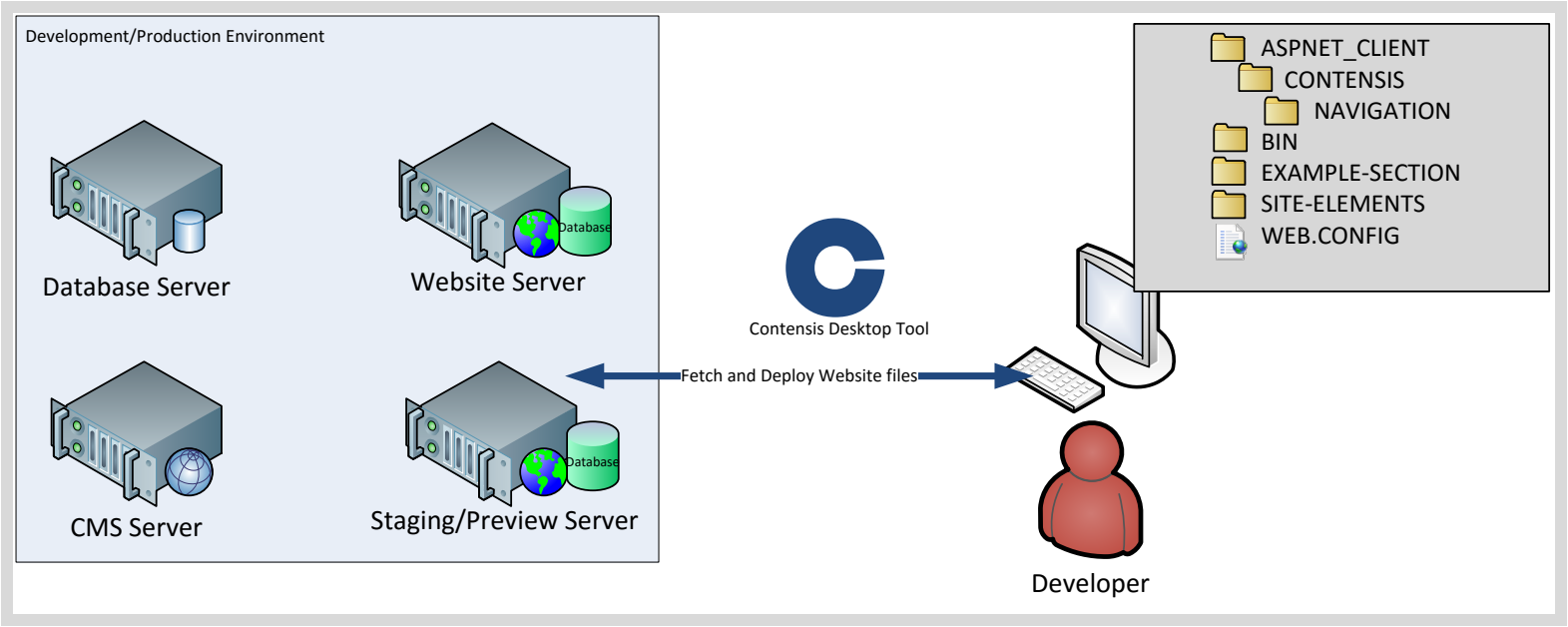
Contensis

Topics Covered

- [ ] Introduction
- [ ] Architecture
- [ ] Your Development Environment
- [ ] Nodes
- [ ] Registering assets and Master Pages
- [x] Relationships
- [ ] Taxonomy & Metadata
- [ ] Item Templates
- [ ] Querying and Searching data
- [ ] Propertied Razor View
- [ ] ClientApi
- [ ] Forms and ApiControllers
- [ ] Localisation and Personalisation
- [ ] Marketplace

Contents

Architecture



Explanation of the Parts of the Site

CMS_XML.xml file

This file contains information such as whether a piece of content is set to be included in menus, its path, title etc...

```
<contentitem id="W11" nodeid="33865" guid="11" typeid="0" type="" version="3297" name="who-we-are" displayname="Who
We Are" menuname="Who We Are" order="4" includeinmenu="1" includeinsearch="1" includeinatoz="1" includeinsitemap="1"
 parent="500" languageid="0" level="1" path="/About-us" url="http://www.company.com/About-us/who-we-are.aspx" size="
0" modifieddate="2015-02-18 10:47:59" isenabledforpublishing="1" ispublished="1" publisheddate="2015-02-18 10:47:59"
 mastercontentid="0" hasshadows="0" templatecontentid="13" template="Standard Page"></contentitem>
```

When a web page first loads, it will try and get key data about itself such as Title and Path from this file so it can be pulled in using the WebAPI. This improves performance by removing the need for a database call. If however the page is published but the CMS_XML.xml hasn't updated, then this data will be retrieved from the database as a fall-back.

Before any code in a Razor View will execute the Razor View itself must exist in the CMS_XML file. This means you need to create the file in the CMS and wait for this file and the CMS_XML to publish before you can download the content through the desktop tool.

CMS_Relationship_XML.xml

This file contains all the data about relationships for the site. As with the CMS_XML file, if you need to retrieve relationship data using our WebAPI, it will retrieve the data from this file.

Your Development Environment

As part of this training you have been given a machine with Microsoft Visual Studio Express 2010 installed and have access to your own sample CMS installation.

Setting up Website Locally

From R7.2, we can make use of the Contensis Desktop Tool to create a local copy of a Contensis site. The tool can be downloaded from here http://support.contensis.co.uk/ClientArea/Desktop-Tool/Contensis-Desktop-Tool.aspx

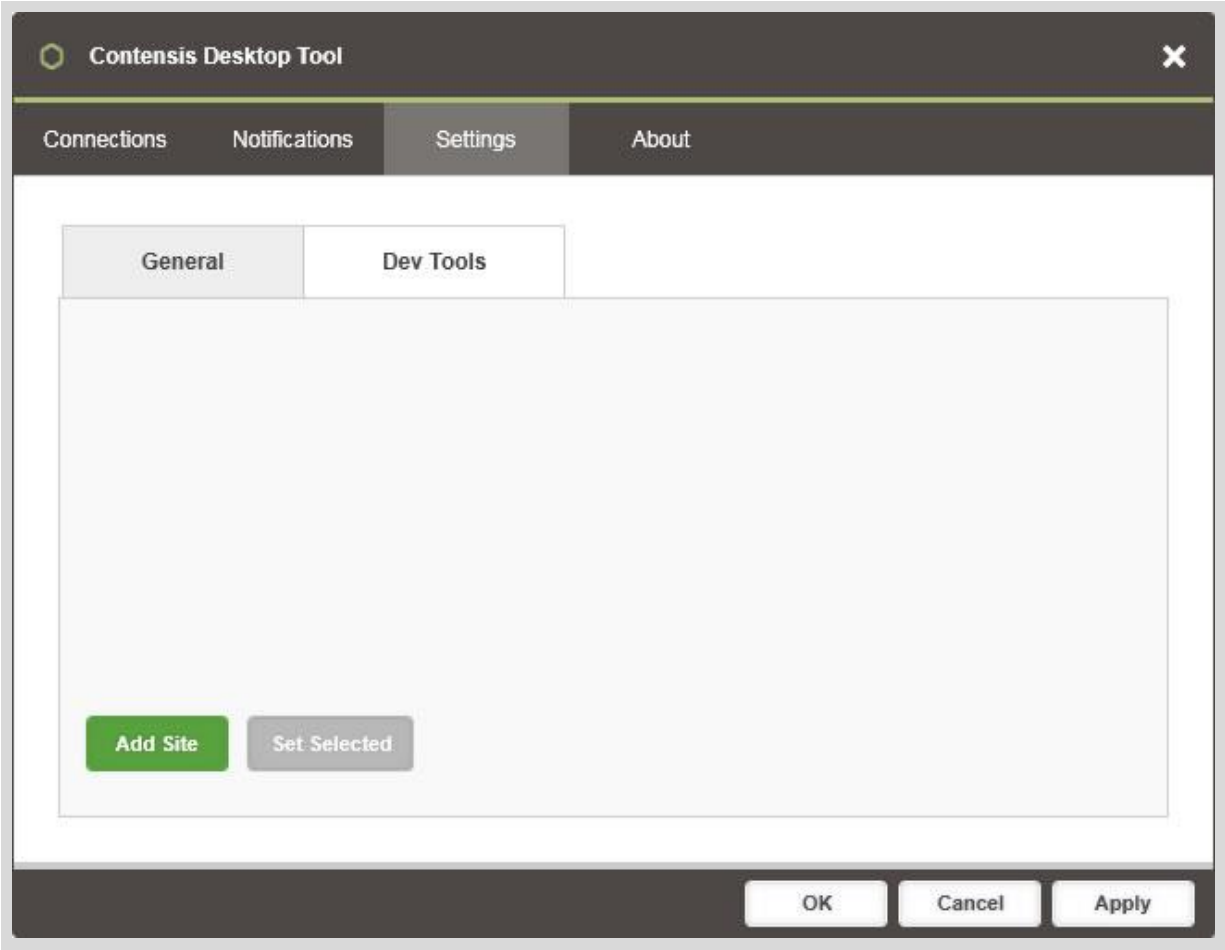
Once you've downloaded and installed the tool, right click on the icon  in the system tray and click on the preferences option.

Next click on the button 

and enter the following Connection Settings:

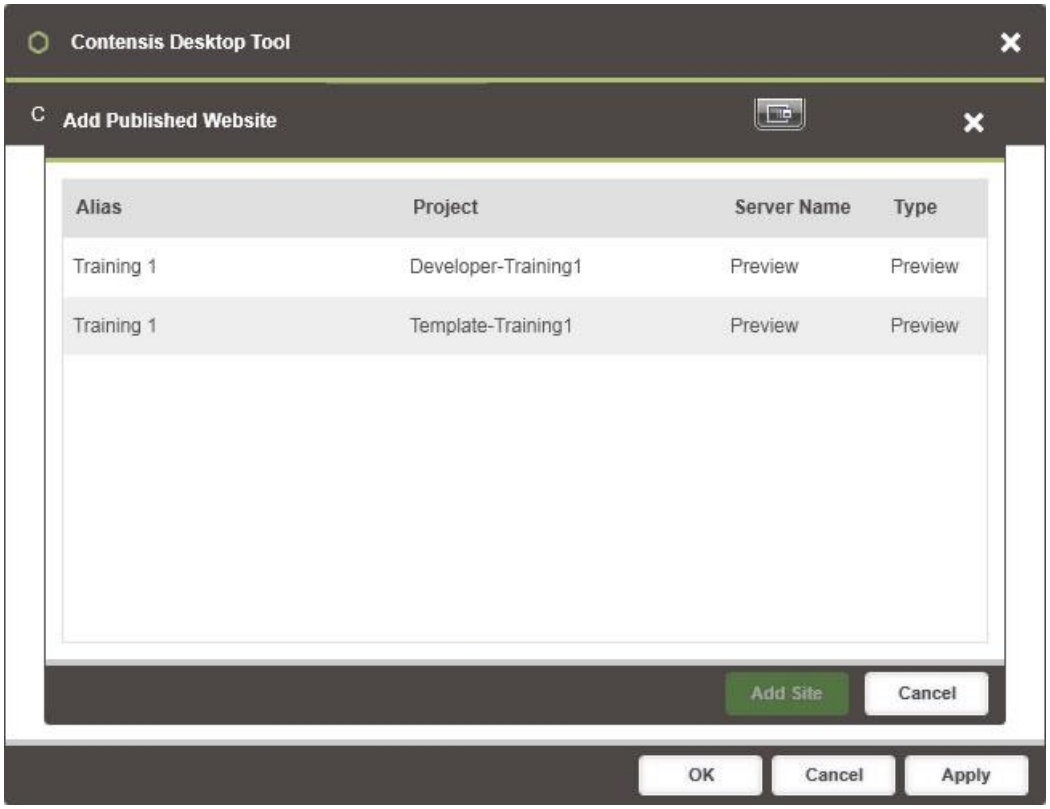
- ☐ Set the Alias as DeveloperTraining
- ☐ The URL will be http://cms.trainingX.contensis.co.uk
- ☐ Username is DevAdmin
- ☐ Password will be trainingX
- ☐ Leave the *Poll interval* set as *10*
- ☐ Ensure Make Default is Checked

The next step is to configure the Developer Tools settings through the desktop tool interface. Click on the *Settings* option and click on the *Developer Tools* tab and you should see the screen below.

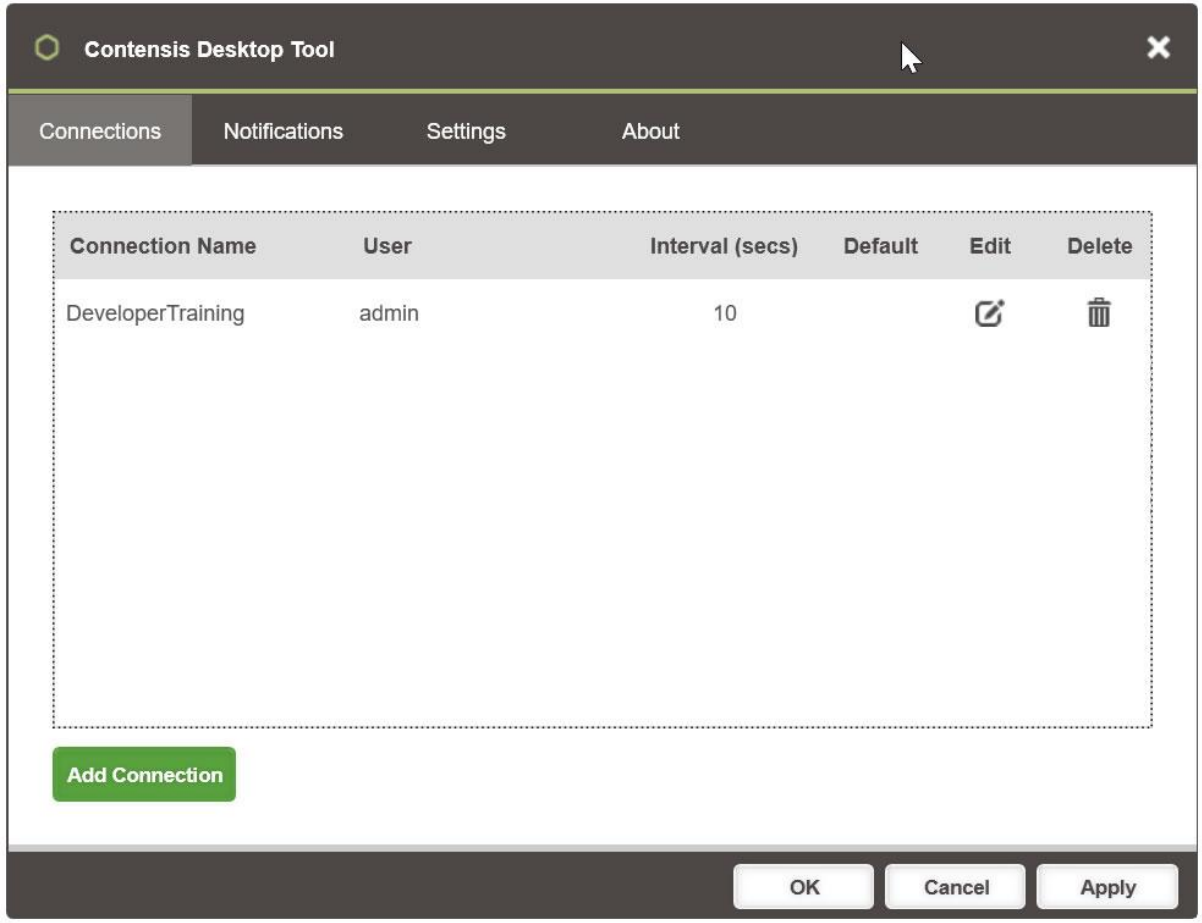


Click on the *Add Site* button and a new dialogue window will appear with the available published websites.

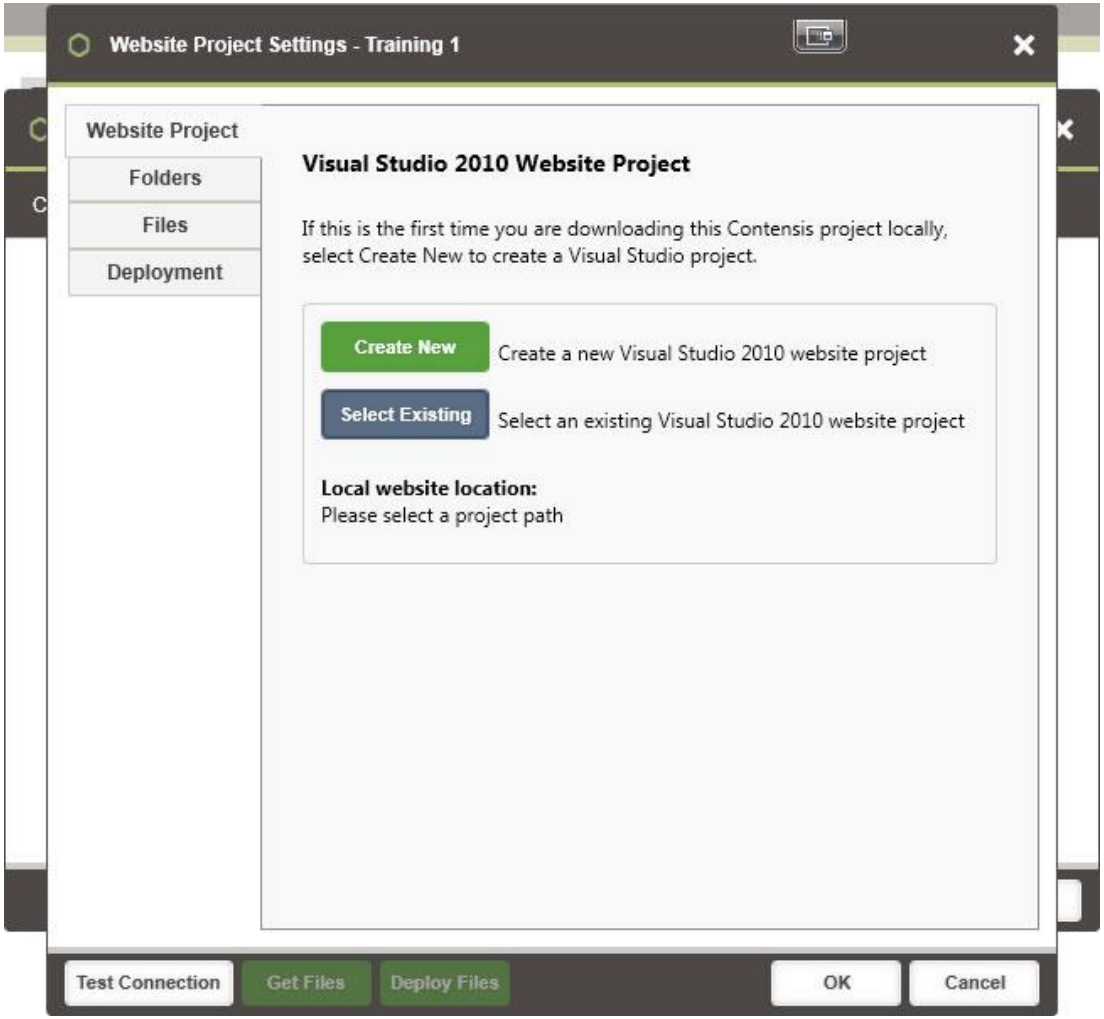
Select the *Developer-Training* project preview website and click *Add Site*.



The next stage is to specify the settings, this is done through the *Website Project Settings*, to initiate this click *Edit* against the *DeveloperTraining* line.



We will first create a visual studio project, so click on the *Create New* option and enter ContensisDeveloperTraining for the Project Name and provide a path of C:\.

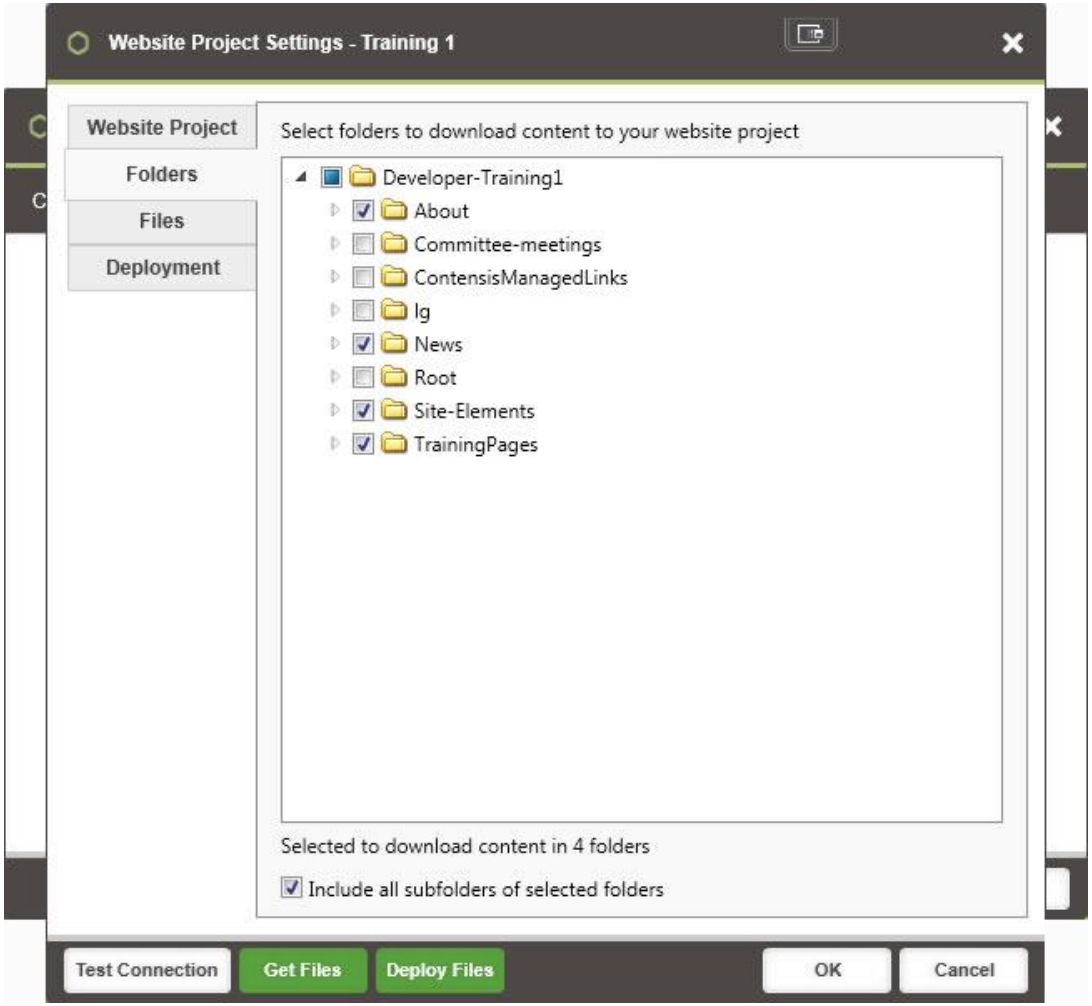


You should get the following confirmation

Next you need to specify the *folders* you require to be fetched from the Contensis published website.

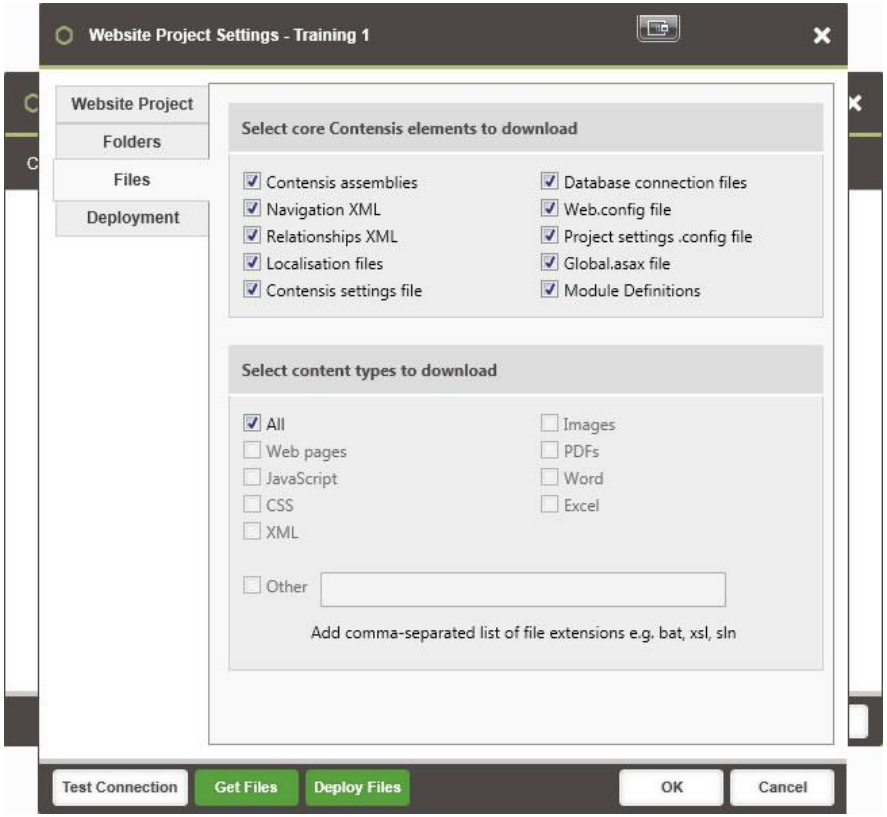
Click on the Folders tab. You will be provided with a view of the folders available, select the *About*, *News*, *TrainingPages* and *Site Elements* folders.

Ensure that the check box *Include all subFolders of selected folders* is checked.



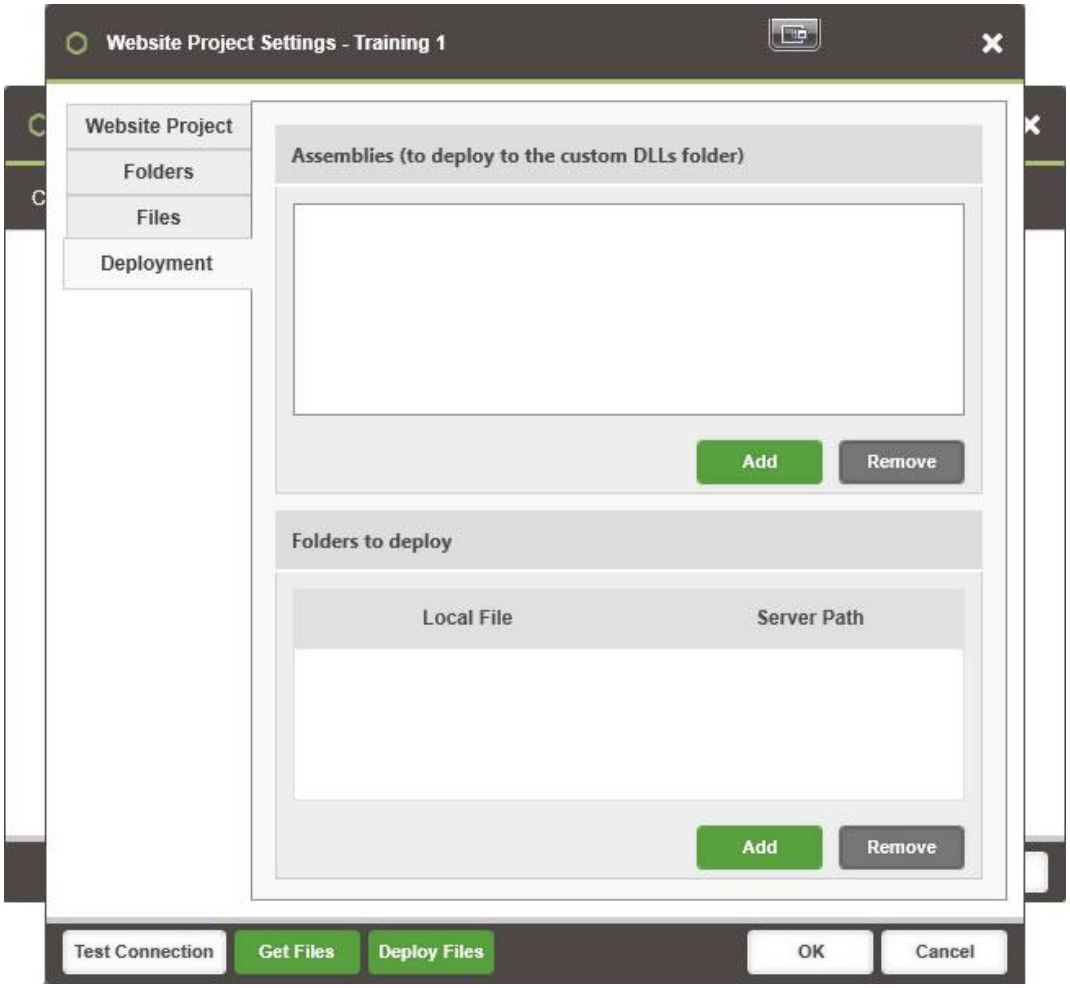
Next click on the *Files* tab, we are not going to change anything today but you change choose what content types you wish to download. This allows you to prevent a large number of PDF documents from being downloaded on to your machine for example.

We recommend that you always leave all the core Contensis elements ticked as all of these will be required for you to run a Contensis site locally.



Now click on the *Get Files* button and click on the OK button when prompted and the files will download.

You can also configure the desktop tool to deploy back to the front end server you downloaded them from. When editing the Website Project Settings, click on the *Deployment* tab.



From this screen you can configure custom dll's to be deployed to the front end server. These will be deployed to the *bin/custom_dlls* folder. Placing them in this folder protects them from being deleted during the Contensis upgrade process.

You can also deploy physical files back to the front end server. To configure this, click the Add button under the *Folders to Deploy* section and set the local path to be your local *Site-Elements* folder and set the remote path to be /Site-Elements/. The remote path is the path on the server.

To deploy the files, right click on the *Desktop Tool* icon in the system tray and click the *Deploy Files* button.

Note: This will only deploy the files to the front end server. If the files need to be deployed to the CMS then this needs to be done through WebDav or if your instance doesn't have a WebDav license then this will need to be done manually.

WebDav

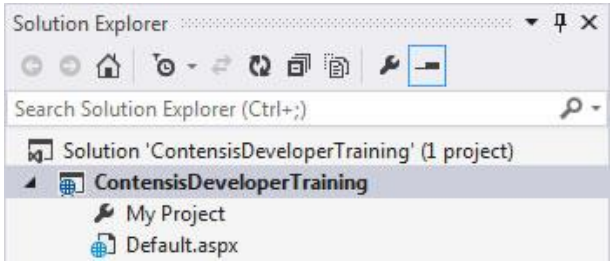
You notice if you now right click on the desktop tool icon you get and option to *Open Contensis Folder*. This indicates our premium WebDav module has been included as part of the licence for the CMS.

This allows you to browse the files in the CMS on your local machine. You can perform workflow actions, update properties and Metadata on content without needing to login to the CMS. You can also open files and edit them on your local machine and have these changes saved to the CMS. We'll cover how this module can improve the Contensis development workflow later in the course.

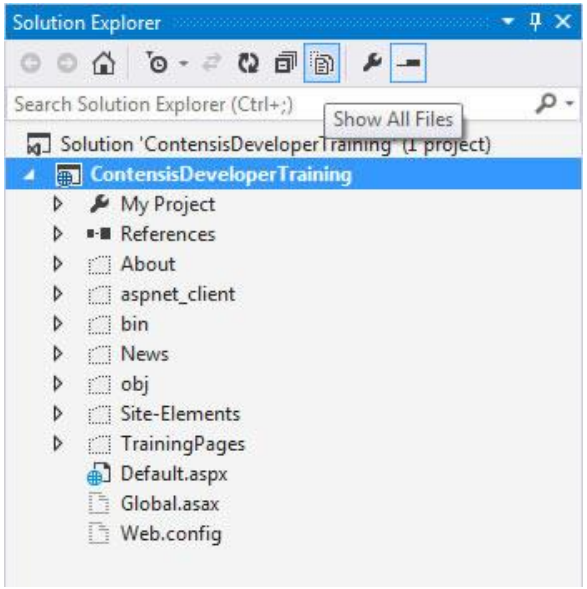
Configuring the Project in Visual web developer

So now we can open the project that the Desktop Tool has created, so OPEN IT??

In the solution explorer you will initially see:



Now click on the *Show All Files* button to reveal the additional files.

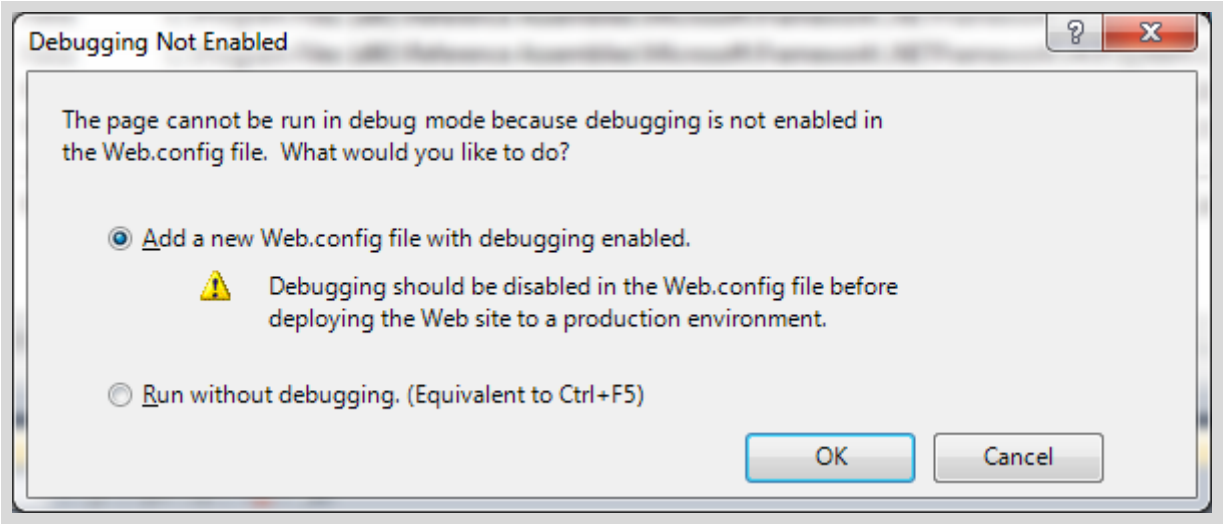


Right click on the *About*, *News*, *TrainingPages* and *Site-Elements* folders and click *Include in Project*. This ensures that when opened, files in these folders will have the correct syntax highlighting applied and where available, intellisense will be enabled.

To test that everything is setup correctly, right click on a page in the *About* folder and choose *View in Browser*. If everything is successful then the page will be rendered exactly as it does on your preview server.

Testing we can Debug our Code

At this stage we should have our local sample fully configured so let's test that we'll be able to debug our code. This time right click on a page in the ***About*** folder and set it as the start-up page. Now hit the *Start Button* and you should see the following prompt



Click OK, and Visual Studio will update the Weg.Config file to run in debug mode.

WebApi Nodes

Current/Content Node

```
@using Contensis.Framework.Web

<h2>Current Node Basic Properties</h2>

<h3>Title: </h3>
<p>@CurrentNode.Title</p>

<h3>Path: </h3>
<p>@CurrentNode.Path</p>

<h3>Uri: </h3>
<p>@CurrentNode.Uri</p>

<h3>Type: </h3>
<p>@CurrentNode.Type</p>

<h3>Filesize: </h3>
<p>@CurrentNode.Size (Kb)</p>
```

```
<h3>Name: </h3>
<p>@CurrentNode.Name</p>

<h3>Include In Settings: </h3>
<p>@CurrentNode.IncludeInSearch</p>
<p>@CurrentNode.IncludeInMenu</p>
<p>@CurrentNode.IncludeInSiteMap</p>
<p>@CurrentNode.IncludeInAtoZ</p>

<h3>First Published Date: </h3>
<p>@CurrentNode.PublishedDateTime</p>
```

CurrentContext

```
@using Contensis.Framework.Web
@{
  //Working with css files
  CurrentContext.Page.CSS.Add("/assets/css/widgetStyle.css");

  //Working with JavaScript files
  CurrentContext.Page.Scripts.RegisterJQuery();
  CurrentContext.Page.Scripts.Add("/assets/css/widgetScripts.js");
  CurrentContext.Page.Scripts.Add("/assets/css/widgetScripts.js", ScriptLocation.BodyEnd);
```

```
 //Metadata
   //Will always render in this format <meta http-equiv="X-UA-Compatible" content="IE=edge">
   CurrentContext.Page.MetaData.Add("X-UA-Compatible", "IE=edge");

   //Localisation
   CurrentContext.Localisations.Get("Website.Widget.Title", CurrentNode.Language);
}
```

Folder Nodes

```
@using Contensis.Framework.Web
<h2>Folder Node Basic Properties</h2>
@{
   FolderNode folderNode = new NodeFactory().Load("/About/Sub-about/SubSubAbout/") as FolderNode;
   <h3>Path: </h3>
   <p>@folderNode.Path</p>

   <h3>Homepage Node: </h3>
   <p>@folderNode.HomePage.Path</p>

   <h3>Parent Folder: </h3>
   <p>@folderNode.Parent.Path</p>
```

```
    <h3>Parent Folder at Specific Depth: </h3>
    <p>@folderNode.Parent.AncestorAtDepth(1).Path</p>

    <h3>Get content of a folder: </h3>
    <ul>
      @foreach (ContentNode page in folderNode.Children()) {
        <li><a href="@page.Path">@page.Title</a></li>
      }
    </ul>
}
```

Exercise 2

Update the /Site-Elements/Razor/FolderList.cshtml Razor View so it does the following:

- ☐ Set a menu title using a HTML heading tag and link it to the current folder homepage.
- ☐ List child pages in the current folder.
- ☐ Use MenuName for link text.
- ☐ For bonus points…
- ☐ Only show pages that have the InlcudeInMenu property ticked.

Relationships

Relationships allow editors to create associations between content in the CMS. For example, you might set up a relationship to allow editors to set related documents for a page.

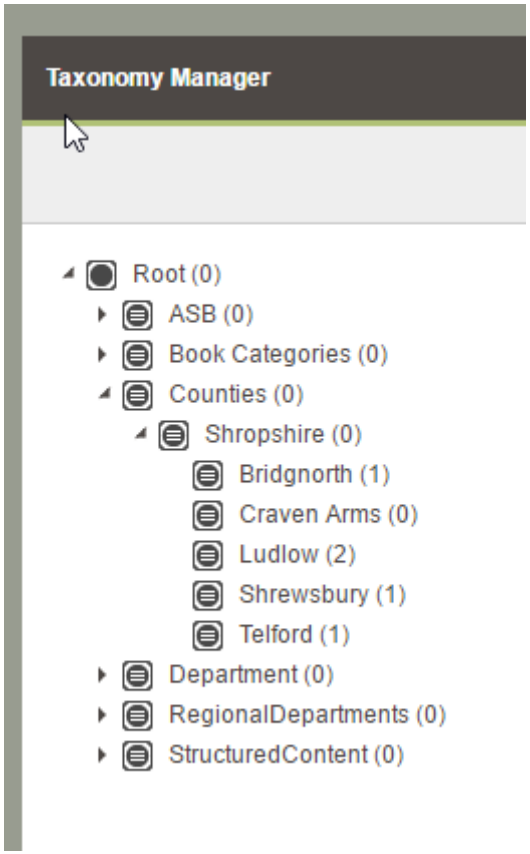
To set up a relationship go to *Management Console* > *Project Setup* > *Relationships* and click the *Create Relationship* button. When the create relationship dialogue appears and set the relation, type and select the Templates and/or Content Types you want to assign the Relationship to.

Related Documents

```
@using Contensis.Framework.Web;
@using System.Collections.ObjectModel;
@{
   ReadOnlyCollection<ContentNode> relatedNodes = CurrentNode.RelatedNodes("RelatedDocuments");
   if (relatedNodes != null)
   {
      <ul>
         @foreach (ContentNode relatedNode in relatedNodes)
         {
            <li><a href="@relatedNode.Path" title="@relatedNode.Title">@relatedNode.Title (@relatedNode.Size)kb</a></li>
         }
      </ul>
   }
}
```

Taxonomy

Taxonomy nodes are managed through the *Taxonomy Manager* in the Contensis management console.

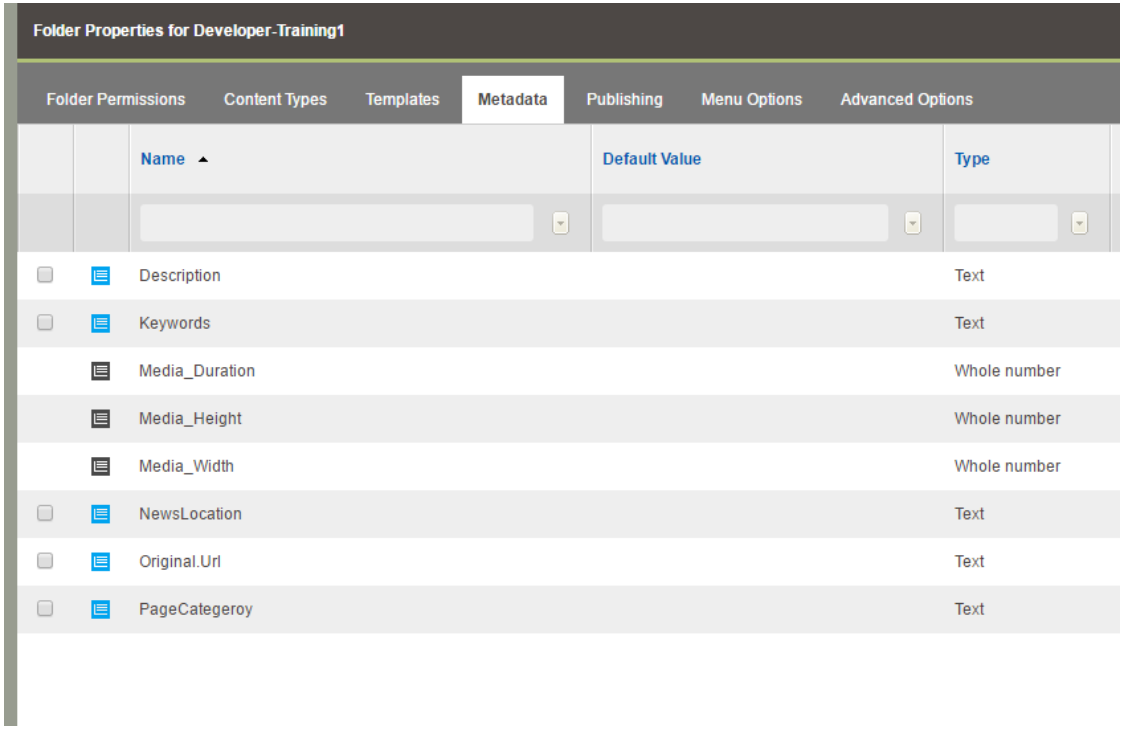


They can be an unlimited number of levels deep and the *Taxonomy Manager* can hold an unlimited number of Taxonomy nodes. Typically, this is used to provide custom categorisation of pages. The next section covers how you can use metadata to tag Contensis content.

Metadata

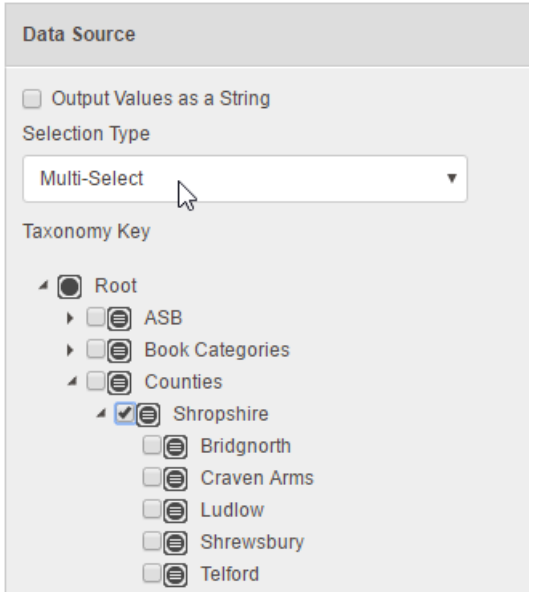
Metadata can be assigned at any folder level though we often create fields on the root node so a field can use by any section of site in the futrue. Metadata can be assigned to any content type in Contensis including webpage and documents.

To assign metadata, *right click* on the *Root* folder and select *Properties* from the menu. Select the *Metadata* tab.



In order to point a metadata field to a Taxonomy node. You need to select the *Contensis Data Source* option in the left hand panel and configure the settings as follows:

- Ensure Output to Database is checked
- Select *Treeview* under the *Display As* panel
- Select Taxonomy as the Data Type
- Set the Selection Type to be Single/Multi Select
- Select the parent taxonomy node you want editors to select the categories from.
- We recommend using the treeview option when working with Taxonomy as it allows editors to go more than one level deep down the Taxonomy tree and provides a more user friendly way for editors to select the data.



Dynamic Data Object

The dynamic data object allows developers to access data stored against content other than the main body of content. This could be custom metadata or structured content data.

Below is an example of how to render this data through our WebApi

```
<p>@CurrentNode.Data.description.ToString()</p>
```

This will render the Metadata description for the page. You don't need to add the MD_ prefix though if you choose to, the data would still be rendered.

```
<p>@CurrentNode.Data.md_description.ToString()</p>
```

Our WebApi doesn't return data as a conventional string when working with the dynamic data. This is so that when working with Structured Content, you get back the raw html if needs be. If you simple need to render the data as plain text to a page then you to add **.ToString()** after the field name as per the example above. **.ToString().**

To allow for error, field names aren't case sensitive but be aware the you can't have two metadata and/or structured content fields with the same field name.

Exercise 3

Create a node in the *Taxonomy Manager* called **PageCategory** and create some child nodes. Next, create a metadata field called **PageCategory** and link this to the node you just created in the Taxonomy Manager. Assign the metadata field to the **Standard** page template and then edit some pages and set some categories. When editing the page, you also need to assign an image as a thumbnail to the page.

Update the **/Site-Elements/Razor/FeaturedContent.cshtml** Razor View so it does the following:

- ☐ Renders featured content from a relationship against the **/TrainingPages/FeaturedContent.aspx** page
- ☐ Each item added should render the following data:
- ☐ Page Category
- ☐ Page Title
- ☐ Page Thumbnail
- ☐ Metadata Description

Razor View Item Templates

A Razor View Item Template allows developers to use standard Contensis listing controls but use different HTML for the individual list items. These are generally used when you want to change the order in which data will render or render custom metadata. An item template means you can do either of these without the need to write custom business logic.

```
@{
  DateTime publishedDate = CurrentNode.PublishedDateTime;
  string articleTitle = CurrentNode.Title;
  <time datetime="@publishedDate">@publishedDate.ToString("dd/MM/yyyy")</time>
  <h3><a href="@CurrentNode.Path" title="@articleTitle">@articleTitle</a></h3>
  <p>@CurrentNode.Data.Description</p>
}
```

Note 1: In the context of an Item Template. CurrentNode refers to the current article or node being added to the page.

Note 2: A full ist of .NET custom date and time format strings can be found here: http://msdn.microsoft.com/en-us/library/8kb3ddd4.aspx

Once we have created our Item Template we then need to add the relevant listing control to the page. This could be the Generic Listing Control or a module specific listing control such as News or Jobs.

Once you've dragged the control onto the page, right click to edit the control properties and scroll down to the Layout section where you'll see a Razor View Item Template property. This opens a Node selector which allows you to select a Razor View to use as the item template.

QueryApi and Search Controls

News listing with thumbnail images using Standard WebApi Query

```
@using Contensis.Framework.Web;
@using Contensis.Framework.Web.Search;
@using System.Collections.ObjectModel;

@{
  IQuery newsArticlesQuery = Query.Where("Property_Type").IsEqualTo("News");
  ReadOnlyCollection<ContentNode> newsArticles = new NodeFinder().Find(newsArticlesQuery, selectCount: 3);

  if (newsArticles.Count > 0)
  {
    <ul>
      @foreach (ContentNode newsArticle in newsArticles)
      {
        DateTime publishedDate = newsArticle.PublishedDateTime;
        string articleTitle = newsArticle.Title;

        <li>
          <img src="@newsArticle.ThumbnailUrl" alt="@articleTitle" />
          <h3><a href="@newsArticle.Path" title="@articleTitle">@articleTitle</a></h3>
```

```
                <span>Published On: </span><time datetime="@publishedDate">@publishedDate.ToString("dd/MM/yyyy")</time>
                <p>@newsArticle.Data.Description</p>
            </li>
        }
    </ul>
  }
}
```

Exercise 4: Book Listing

Update the /Site-Elements/Razor/BookListing.cshtml Razor View so that it displays a list of books with the following data.

- ☐ Title (People should be able to click through to record page)
- ☐ Strapline
- ☐ Book Description
- ☐ Price
- ☐ ISBN
- ☐ Thumbnail

Search Controls

Weighted Search

```
IQuery pageQuery = Query.WhereFullTextContains("Contensis", In.Column("Property_Title", 100), In.Column("SC_Content", 10)).And("Property_CT_ID").IsEqualTo("0");
ReadOnlyCollection<ContentNode> pages = new NodeFinder().Find(pageQuery, selectCount:3);
```

The example below makes use of standard .NET html helpers for form fields, and whilst this is our recommended approach. The search controls in the next section will all work if the fields are entered as normal HTML form fields in the Razor View.

Whilst the example shows getting data from the posted back fields, it is possible to get data from the query string. This involves writing additional without providing any real benefit to the end user. Theses is an example of working with the querystrings on ZenHub.
https://zenhub.zengenti.com/Contensis/R83/Development/Razor/Workingwithquerystrings.aspx

News Search

```
@using Contensis.Framework.Web
@using System.Collections.Specialized
@{
   //Check if we have search paaemeters the set these as the default values for the relevent search fields
   string searchStringKeywords = !string.IsNullOrEmpty(Request.Form["keywords"]) ? Request.Form["keywords"] : string.Empty;
   string searchStringCategory = !string.IsNullOrEmpty(Request.Form["categoryFilter"]) ? Request.Form["categoryFilter"] : string.Empty;
}
```

```
<h3>News Search</h3>
@Html.Label("Keywords", "keywords")
@Html.TextBox("keywords", searchStringKeywords, new { id = "keywords", placeholder = "Enter keywords" })

@{
   TaxonomyNode newsCategories = CurrentContext.Taxonomy.GetByPath("/StructuredContent/NewsCategories/");
   newsCategories.SortOrder = TaxonomySortOrder.Alphabetical;

   Html.Label("Category Filter", "categoryFilter");
   if (newsCategories != null)
   {
      @Html.DropDownList("categoryFilter",
      "Please Select",
      newsCategories.Children.Select(nC => new SelectListItem()
      {
         Selected = nC.Key == searchStringCategory ? true : false,
         Text = nC.Value,
         Value = nC.Key
      }))
   }
}

<input type="submit" id="submit" name="submit" value="Search News" />
```

Exercise 5: Book Search Control

Update the /Site-Elements/Razor/BookSearch.cshtml Razor View so that it displays a search control with the following search fields:

- ☐ Textbox field
- ☐ Dropdown filter for either book genres or publishers
- ☐ If a user clicks the search button then on reload, the fields should be pre-populate with the search filters the user as entered or selected

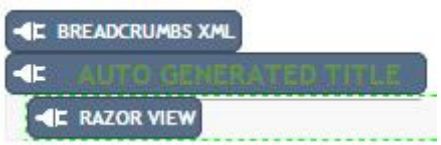
Propertied Razor Views

Introduction

Propertied Razor Views allow editors to change to look or behaviour of a control without needing to edit the code itself. These have many other uses from simply giving instructions to writing Node Queries.

Instruction Element

When you drag a standard Razor View into a placeholder or template, it display's a component with the title of Razor View



This isn't very useful when you have multiple Razor Views on screen that have either been dragged in or are part of the main page or base template.

To help make it easier for developers and editors to quickly get an idea of what a Razor View does, we can convert it to a Propertied Razor View without needing to add any editable properties.

When editing a Razor View in Contensis, you have an additional tab called Configuration, which is where you write the code to define any properties and Node Queries for the Razor View.

Code for Configuration Tab

```
<control name="DT - Folder List" showInMenu="true" category="Navigation" viewingGroup="1">
  <properties>
    <category name="Information">
      <instruction label="This Razor View will list all content for the current section" />
    </category>
  </properties>
</control>
```

Looking at above code the value for the name attribute in the control tag will be displayed when the Razor View is dragged into a page or template. We recommend that you prefix the control name with an abbreviated version/acronym of your organisation name so you can distinguish between custom controls and standard Contensis controls when looking at the controls in the CMS editor or context menu.

The string in the label attribute on the instruction tag will display when an editor or developer right clicks on the Razor View and clicks web control properties or when the Razor View is first dragged onto a page or template. The purpose of the instruction element is to give someone a general overview of what data the Razor View will render.

Node Queries

In a previous example, we queried the database for news articles using these two lines of code.

```
IQuery newsArticlesQuery = Query.Where("Property_Type").IsEqualTo("News");
ReadOnlyCollection<ContentNode> newsArticles = new NodeFinder().Find(newsArticlesQuery, selectCount: 3);
```

The example below shows how we retrieve the same data but using a node query instead.

It is also possible to pass search filters into a node and in the previous example and the example below, you can see we have utilised the orderby attribute on the <nodeQuery /> property.

Code for Configuration Tab

```
<control name="News Listing Using Node Query" showInMenu="true" category="Dev Training" viewingGroup="1">
  <properties>
    <nodeQuery name="NewsArticlesQuery" orderby="Property_DatePublished desc">
      <where property="Type" operator="IsEqualTo" value="News" />
      <and property="Title" operator="Contains" value="@Request.Form.keywords" />
      <and property="Data.TaxonomyCategories" operator="Contains" value="@Request.Form.categoryFilter" />
    </nodeQuery>
  </properties>
</control>
```

Below is the update code for the Razor View

```
ReadOnlyCollection<ContentNode> newsArticles = Properties.newsArticlesQuery;
```

You'll notice that we used *Properties.* to reference the query and we called newsArticlesQuery, which the name attribute value on the main <nodeQuery> property.

The rest of our listing code can remain the same.

We' recommend that you utilise Node Queries for building simple listing controls. For listing controls that require more complex filtering, we'd recommend you utilise our standard WebApi query interface. Further documentation and examples can be found in the following ZenHub Articles: https://zenhub.zengenti.com/Contensis/R83/Development/Razor/Workingwithquerystrings.aspx and https://zenhub.zengenti.com/Contensis/R83/kb/building-a-website/Search/creating-a-search.aspx

Exercise 6: Putting it all together

Update the /Site-Elements/Razor/BookListing.cshtml Razor View so that our book listing control now does the following:

- ☐ The Razor View as a title of BOOK LISTING in the CMS editor and appears in the Dev Training category in the Web Controls context menu
- ☐ Uses a NodeQuery to query the book data from the database
- ☐ Display the books in Alphabetical order by title
- ☐ Filters the books where the title contains the keywords and/or the publisher or genre match the value selected by a user

Standard Properties

As well as Node Queries, we have developed an additional set of properties that allow editors to change the look or behaviour of a Razor View without someone needing to modify the existing Razor View. Below is some example XML with examples of all these properties with the exception of the <repeater> property.

```
<control name="Properties Examples" viewingGroup="1">
  <properties>
    <category name="Basic Properties">
```

```
        <text name="singleline" label="Singleline" />

        <text name="multiline" label="Multiline">
          <parameters>
            <parameter name="PlaceholderText" value="Enter some text here" />
            <parameter name="Multiline" value="true" />
          </parameters>
        </text>

        <text name="list" label="List" editor="single">
          <options>
            <option label="Option 1" value="1"></option>
            <option label="Option 2" value="2"></option>
          </options>
        </text>

        <number name="recordsToShow" label="Records to Show" required="true" description="Specifiy the number of records you the control
to return" default="3" />
</category>

    <category name="Node Picker">
      <node name="image" label="Image">
        <parameters>
          <parameter name="ContentTypes" value="2" />
          <parameter name="ImageWidth" value="100" />
```

```
            <parameter name="ImageHeight" value="200" />
          </parameters>
        </node>
      </category>

      <category name="Taxonomy Selector">
        <taxonomy name="taxonomySelector" label="Taxonomy Selector" editor="multipleselect">
          <parameters>
            <parameter name="RootNode" value="/Counties/Shropshire" />
          </parameters>
        </taxonomy>
      </category>

      <category name="New Options">

        <dateTime name="myDateTime" label="DateTime" />
        <tag name="myTag" label="Tag" />

      </category>
    </properties>
  </control>
```

To reference a property in a Razor View you need to do @Properties.propertyNameAtributeValue e.g @Properties.singleline.

Be aware that the WebApi returns data entered using the <number> property as type double as opposed to integer. This to account for situations where editors may need to enter numbers with decimals. An example would be a map control where editors are required to enter longitude and latitude co-ordinates.

This line of code will cover the property to an integer.

```
int articlesPerPage = Convert.ToInt32(Properties.articlesPerPage);
```

Repeaters

The repeater property allows developers to define a group of properties that editors can set more than once. Any property can be repeated and the minimum and maximum number of sets can be defined.

Configuration

```
<control name="Slideshow" showInMenu="true" category="Dev Training" viewingGroup="1">
  <properties>
    <repeater name="slides" label="Slides" description="Select slides for slideshow" min="1" max="3">
      @*Properties go here*@
    </repeater>
  </properties>
</control>
```

Content

```
@using Contensis.Framework.Web
@foreach (var slide in Properties.slides)
{
```

```
    //Render content here
}
```

When looping through a repeatable set, you need to define the value in the **name** attribute of the <**repeater**> property as the array to loop through. If you use a node or taxonomy selector, then the property as the relevant node type. This allows developers to utilise standard WebApi properties such as **Title** and **Path** without needing to convert the data to a different object or type. See an example below.

```
<a href="@slide.propertyName.Path"> @slide.slideTitle</a>
```

Where working with an image node, you return the alt text via the **MenuName** property. If you only have the path of an image, then you this to load the individual image node as a **ContentNode** as per the example below:

```
ContentNode imageNode = new NodeFactory().Load("/Site-Elements/images/NewsSamples/BankofEngland.jpg") as ContentNode;
```

You could pass in a dynamic value if say you want the thumbnail image node for item is a list e.g **Load(listItem.ThumbnailUrl)**

Exercise 7: Slideshow Control

Modify the **/Site-Elements/Razor/Slideshow.cshtml** so that it does the following.

- ☐ Displays with the name **SLIDESHOW** when dragged into the editor
- ☐ Has a repeater that contains the following properties:
- ☐ A node picker restricted to just pages
- ☐ A node picker restricted to just images
- ☐ A slide slide title property
- ☐ A slide strapline field
- ☐ Editors must add one slide and they can add no more than 3 slides

Once the configuration has been updated, the code/content will need updating so that each slide should render the following data.

- ☐ The image path
- ☐ The image alt text
- ☐ A link wrapped around the image
- ☐ The slide title
- ☐ The slide strapline

The cycle2 plugin has already been referenced in the Razor View and the slideshow should start automatically without the need for any additional JavaScript. This example should help with rendering the data http://jquery.malsup.com/cycle2/demo/overlay.php.

There is **Slideshow.aspx** page in the **/TrainingPages** folder that you add the Razor View to to test.

Client API

The Contensis ClientAPI allows developers to retrieve Contensis data via our secure RESTful web services. You can also call this service securely from a website that isn't powered by Contensis. We return the data as JSON to give you the flexibility to handle and manipulate the data as you see fit.

The examples below show how you can build a predicitive search use a couple of query methods. It also possible to return Taxonomy and Relationship data through our ClientAPI.

Example 1 – Live Search Standard Query

liveSearch-StandardQuery.js

```
$(function () {
    $searchKeywordsInput = $('.searchKeywordsInput');
    $searchResults = $('.searchResults');
    $resultsCount = 6;

    $searchKeywordsInput.attr('autocomplete', 'off').keyup(function (event) {
        if (event.keyCode > 40 || event.keyCode == 8 || event.keyCode == 9) {
            $searchResults.empty();
            $searchResults.append("<li class='loading'><img src='/Site-Elements/images/ajax-loader.gif' />");
            $searchResults.show();
            var searchResultsQuery =
Contensis.Search.Query.where("Property_Title").contains($searchKeywordsInput.val()).and("Property_Path").startsWith("/About");
            var searchResults = Contensis.Search.NodeFinder.find(searchResultsQuery, { start: 0, resultCount: $resultsCount }, function (result) {
                $searchResultsCount = result.TotalMatches;
                if($searchResultsCount  > 0) {
                    $searchResults.empty();
                    if($searchResultsCount == 1) {
                      var searchResult= result.Nodes[0];
                      $searchResults.append('<li><a href="'+ searchResult.Path +'" title="Go to '+ searchResult.Title+'">'+
searchResult.Title+'</a></li>');
                    }
                    else
```

```
                {
                  for (var i = 0; i < $searchResultsCount; i++) {
                     var searchResult= result.Nodes[i];
                     $searchResults.append('<li><a href="'+ searchResult.Path +'" title="Go to '+ searchResult.Title+'">'+
searchResult.Title+'</a></li>');
                  }
               }
            }
            else
            {
               $searchResults.empty();
               $searchResults.append('<li>There are no results!</li>');
            }
         });
      } else if (event.keyCode == 27) {
         //User pressed escape key.
         $searchResults.hide();
      }
   }).keyup(function (event) {
      if (event.keyCode == 13) {
         //User pressed enter key.
         $searchResults.hide();
      }
   });
});
```

liveSearch-StandardQuery.cshtml

```
@using Contensis.Framework.Web;
@{
        CurrentContext.Page.Scripts.RegisterJQuery();
        CurrentContext.Page.Scripts.RegisterContensisApi();
        CurrentContext.Page.Scripts.Add("/SiteElements/js/liveSearch-StandardQuery.js");
}
@Html.Label("Keywords", "keywords")
<input type="text" id="keywords" class=”standardKeywordsInput”  name="keywords" placeholder="Enter keywords" />
@Html.TextBox("keywords",  new { class = " standardKeywordsInput ", id = " standardKeywordsInput ",  placeholder = "Enter keywords" })
<input type="submit" id="submit" name="submit" value="Search Site" />

<ul id=" standardSearchResults ">
</ul>
```

Example 2 – Live Search Full Text Query

```
$(function () {
   $searchKeywordsInput = $('.searchKeywordsInput');
   $searchResults = $('.searchResults');
   $resultsCount = 6;

   $searchKeywordsInput.attr('autocomplete', 'off').keyup(function (event) {
      if (event.keyCode > 40 || event.keyCode == 8 || event.keyCode == 9 ) {
```

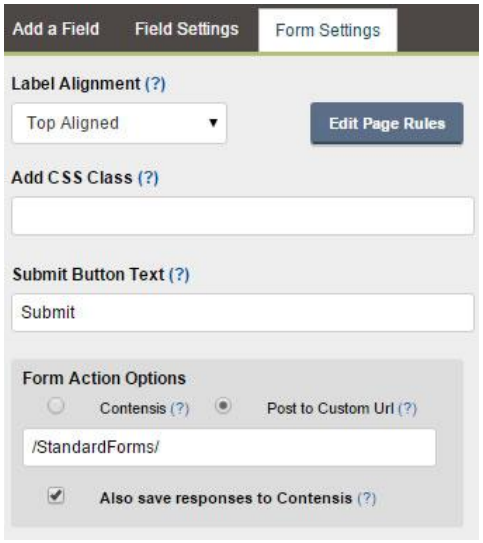
```
        $searchResults.empty();
        $searchResults.append("<li class='loading'><img src='/Site-Elements/images/ajax-loader.gif' />");
        $searchResults.show();
        var searchResultsQuery = Contensis.Search.Query.whereFullTextContains($searchKeywordsInput.val(),
          [
           {columnName: 'Property_Title', weight: 100},
           {columnName: 'SC_Content', weight: 10}
           ]).and("Property_IncludeInSearch").isEqualTo("1");
        var searchResults = Contensis.Search.NodeFinder.find(searchResultsQuery, { start: 0, resultCount: $resultsCount }, function (result) {
          $searchResultsCount = result.TotalMatches;
          if($searchResultsCount > 0) {
            $searchResults.empty();
            if($searchResultsCount == 1) {
              var searchResult= result.Nodes[0];
              $searchResults.append('<li><a href="'+ searchResult.Path +'" title="Go to '+ searchResult.Title+'">'+
searchResult.Title+'</a></li>');
            }
            else
            {
              for (var i = 0; i < $searchResultsCount; i++) {
                var searchResult= result.Nodes[i];
                $searchResults.append('<li><a href="'+ searchResult.Path +'" title="Go to '+ searchResult.Title+'">'+
searchResult.Title+'</a></li>');
              }
            }
```

```
                }
                else
                {
                    $searchResults.empty();
                    $searchResults.append('<li>There are no results!</li>');
                }
            });
        } else if (event.keyCode == 27) {
            //User pressed escape key.
            $searchResults.hide();
        }
    }).keyup(function (event) {
        if (event.keyCode == 13) {
            $searchResults.hide();
        }
    });
});
```

Form Helpers and Api Controllers

Introduction

Contensis Form helpers allow you to create your own forms using Razor Views. It is also possible to add validation to these forms and control where the validation summary appears. The data for these forms can be posted to a custom ApiController so you can have full control over what happens to the data.



It is also possible to post standard Contensis Forms to an ApiController. This is done by updating the *Form Actions Options* setting under the *Form Settings* tab to *Post to Custom Url* as per the example below. You need to set the url based on the controller class name you wish to target.

Posting data from a standard Contensis form

StandardFormApiController.cs

```
using System;
using System.Collections. Specialized;
using System.Net;
using System.Net.Http;
using System.Net.Http.Formatting;
using System.Web.Http;
using Contensis.Framework.Web.Controllers;

public class StandardFormsController : ContensisFormsApiController
{
 protected override HttpResponseMessage Post(NameValueCollection formValues)
 {
    return CreateResponse("", HttpPostResponseAction.Redirect, new HttpRedirectOptions("GET", "/Thank-You.aspx?name=" +
formValues["Name"]));
 }
}
```

When you're working with data from a standard Contensis form, you need to reference the Label name as defined in the CMS for the relevant form field. This is because on the front end, Contensis generates the name attribute value dynamically for validation purposes.

Exercise 8: Custom controller

Create a new form in the */Site-Elements/Forms* folder that contains the following fields:

- ☐ Name
- ☐ Email
- ☐ Department
- ☐ Enquiry Details

Ensure the form does the following:

- ☐ Saves the data to Contensis
- ☐ Posts the data to a custom controller
- ☐ The controller needs to do preform some logic to return a different email address based on the department selected.
- ☐ Return a message that says the following:
- ☐ Thank you *Name* for your enquiry, your enquire details are *Enquiry Details* and your enquiry has been sent to *Department Email* address.

You can drag the form onto the **/TrainingPages/EnquiryForm.aspx** page to test.

Custom Forms

As well as standard Contensis forms, it also possible to write your own custom forms using Razor Views. It is possible to attach validation to a custom form using a combination of the jquery validate plugin and standard .NET form helpers. The data submitted by these forms won't be save to Contensis but is yours to do with as you see fit.

The submission of the forms is done but utilising custom controller and http routing in .NET. Further information on this can be found here http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-apiFeedback-Form.cshtml

```
@using Contensis.Framework.Web
@{
   CurrentContext.Page.Scripts.Add("http://ajax.aspnetcdn.com/ajax/jquery.validate/1.13.1/jquery.validate.min.js");
   CurrentContext.Page.Scripts.Add("http://ajax.aspnetcdn.com/ajax/mvc/5.1/jquery.validate.unobtrusive.min.js");

        //Set some validation
        Validation.RequireField("firstname", "Firstname is required");
        Validation.RequireField("surname", "Surname is required");
        Validation.Add("email",
                Validator.Required("You must provide a valid email address"),
        Validator.Regex(@"^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$", "Invalid format for an email address")
        );
        Validation.RequireField("enquiry ", "Enquiry details are required");
}

@using (Html.BeginContensisForm("SendFeedback", "api/Custom/Feedback"))
{
  @Html.ValidationSummary("", new { @class = "validation-summary-class" })
  <p>
    @Html.Label("Firstname","firstname")
    @Html.TextBox("firstname", null, new { id="firstname", @class = "form-control" })
  </p>
```

```
    <p>
      @Html.Label("Surname", "surname")
      @Html.TextBox("surname", null, new { id = "surname", @class = "form-control" })
    </p>

    <p>
      @Html.Label("Email", "email")
      @Html.TextBox("email", null, new { id = "email", @class = "form-control", type = "email" })
    </p>

    <p>
      @Html.Label("Enquiry", "enquiry")
      @Html.TextArea("enquiry", null, new { id = " enquiry", @class = "form-control form-textarea", cols = "20", rows="5" })
    </p>
    <input type="submit" name="submit" value="Submit Feedback" />
}
```

FeedbackApiController.cs

```
using System;
using System.Collections.Specialized;
using System.Net;
using System.Net.Http;
using System.Net.Http.Formatting;
using System.Web.Http;
```

```
using Contensis.Framework.Web.Controllers;

[RoutePrefix("api/Custom/Feedback")]
public class FeedbackApiController : ContensisApiController
{
   [HttpPost]
   [Route("SendFeedback")]
   public HttpResponseMessage SendFeedback(FormDataCollection data)
   {
      return CreateResponse("", HttpPostResponseAction.Redirect, new HttpRedirectOptions("GET", "/Thank-You.aspx"));
   }


}
```

Personalisation

Populo

Populo is our new platform that focuses on Customer Experience Management. Populo gathers data about visitors to your Contensis site such where they are and also what their browsing habits/behaviours are. This platform is completely independent of Contensis whoever we have developed a ContactsApi that allows to pull and utilise data from Populo in your Contensis website.

Further information on both Populo and the ContactsApi, along with examples, can be found on ZenHub
https://zenhub.zengenti.com/Contensis/R83/kb/browse.aspx?category=populo

Below is a modified version of a banner control that will show a different banner depending on a user's location.

Slideshow Control Personalised

Configuration

```
<control name="Slideshow Personalised" showInMenu="true" category="Dev Training" viewingGroup="1">
  <properties>
    <repeater name="slides" label="Slides" description="Select slides for slideshow">
      <node name="image" label="Image">
        <parameters>
          <parameter name="ContentTypes" value="2"></parameter>
        </parameters>
      </node>
      <text name="slideTitle" label="Slide Title"></text>
      <text name="slideStrapline" label="Slide Strapline"></text>

      <taxonomy name="country" label="Country" editor="tree">
        <parameters>
          <parameter name="RootNode" value="/Countries/" />
        </parameters>
      </taxonomy>
    </repeater>
  </properties>
</control>
```

```
@using Contensis.Framework.Web
@using Zengenti.Populo;

@{
   //Register assets
   CurrentContext.Page.Scripts.RegisterJQuery();
   CurrentContext.Page.Scripts.Add("/Site-Elements/js/cycle.js");

   CurrentContext.Page.CSS.Add("/Site-elements/css/System.css");

   //Set users location
   var currentContact = Contacts.Current();
   string currentLocation = currentContact.Location.Current != null ? currentContact.Location.Current.Country : "United Kingdom";
}


<div class="cycle-slideshow"
    data-cycle-fx="scrollHoriz"
    data-cycle-timeout="5000">

   <div class="cycle-overlay"></div>
```

```
    @foreach (var slide in Properties.slides)
    {
        if (slide.country.Value == currentLocation)
        {
            <img src="@slide.image.Path" alt="@slide.image.MenuName" data-cycle-title="@slide.slideTitle" data-cycle-desc="@slide.slideStrapline"
/>
        }
    }

</div>
```

Placeholder Data

In Contensis 8.3, It is also possible to manipulate placeholder data. This could be based off data from Populo or other data that is stored in you Contensis website. Below is an example that shows updating an image in a placeholder based on the user's location.

```
@using Contensis.Framework.Web
@using Zengenti.Populo;
@{
    //Set users location
    var currentContact = Contacts.Current();
    string currentLocation = currentContact.Location.Current != null ? currentContact.Location.Current.Country : "United Kingdom";

    //Set hero imagepath
    string heroImagePath = "/ContentLibrary/bannerImage/heroimage-" + currentLocation.ToLower().Replace(" ", "-") + ".jpg";
```

```
        //Update placeholder data
        CurrentContext.Page.Placeholders["heroImage"].Value = "<img src='"+ heroImagePath +"' alt='Hero Image' />";

    }
```

The example above will only work for where the publishing server is set Legay ASPX mode. If the publishing server is set to Master Pages mode, then the placeholder will need to be prefixed with L1_, L2_, L3_ etc.... The level is based on which template the placeholder sits within, typically, placeholders in the Base template will be prefixed with L1. Placeholders in a page template will be prefixed with L2 and placeholders in sub templates will be prefixed will L3.

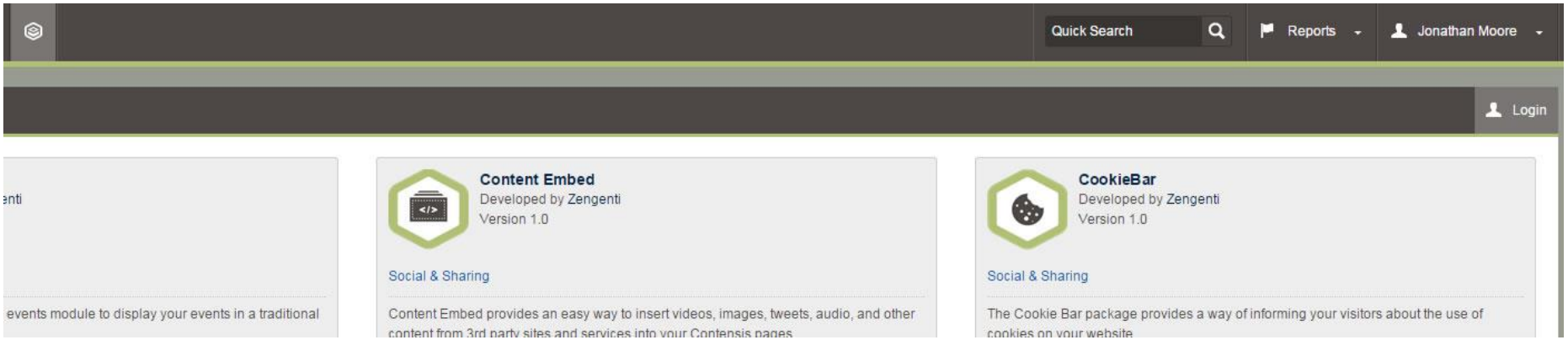
Marketplace

Overview

The Cotensis marketplace allows developers to share a piece of functionality they have developed within Contensis. They can chose to only share thus within their organisation. Alternatively, they can make it available for all Contensis users. This could be a new custom Razor listing, a set of templates or even a piece of structured content.

Developers who want to upload and install packages will need to register by going to https://zenhub.zengenti.com/Contensis/R82/kb/marketplace/Marketplace-Overview.aspx or by contacting their Marketplace administrator.

Once your marketplace account is setup you can navigate to the marketplace by clicking on the icon at the top left of the contensis dashboard. Login to the marketplace using the link at the top right of the marketplace window.



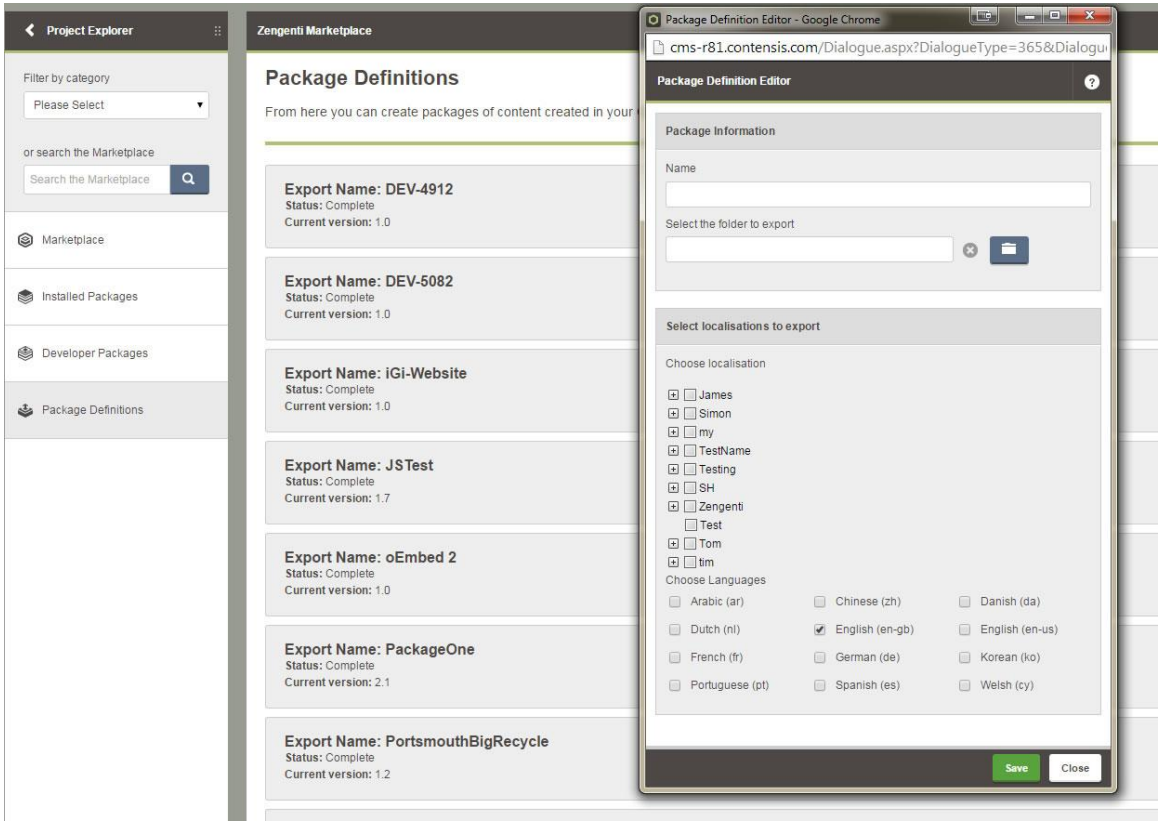
Login to the Marketplace using the following details

Username: TrainingUserX

Password: trainingX

Creating a Package

Once logged in, if you have developer or administrator access, you can click on *Package Definitions* in the left navigation panel.



Click on the green *Create New Definition* button and the *Package Definition* screen will appear.

Give your package a name and select the content within the CMS that you wish to be included in the package by clicking on the blue *Browse* button.

You can also export localisation namespaces and select which languages you want to export.

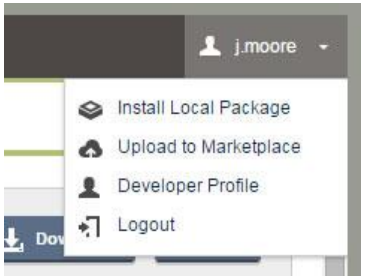
Once you've selected everything you want to export click *Save*. You now have a package definition with the correct files chosen. Select this definition and click on the green *Generate* button.

This will give you a confirmation screen, click OK. The package will now be queued for generation.

Exporting & Installing a Package

Once your package has generated, you should now see a download button next to it. Clicking this will download the package as a .zip file and save it to your machine.

You can then login into another CMS, login to the marketplace and install the package by clicking on your username in the marketplace window and selecting *Install Local Package*. Choose the package you want to install and which folder in the CMS you want to import the files to and click *Install*. You can also choose if you want publish the files immediately or not.



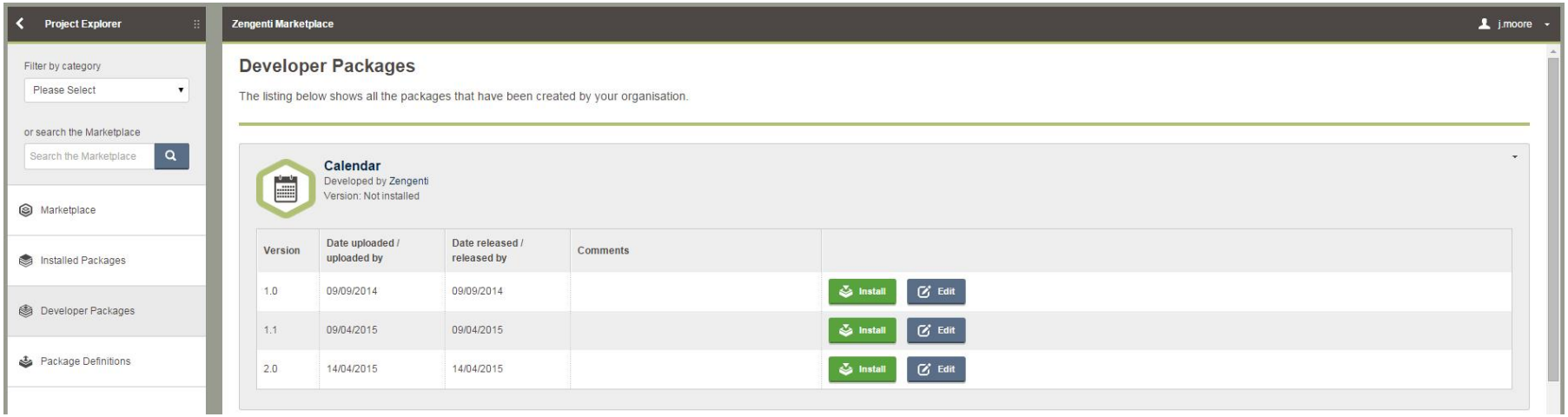
This allows you to move content between any Contensis instance without need to logon to a server. Say for instance you've developed a new set of templates and Razor views on your development instance. Rather than replicating all this manually in your UAT and live instances, you can simply move the files via the marketplace.

Developer Packages

Developer packages are packages that have been shared internally within your organisation or published to the marketplace for all Contensis customers to download and install.

To create a developer package you need to click on your username in the marketplace window and select *Upload to Marketplace*. You can either select an existing package or upload a package definition you've created elsewhere and downloaded. A developer can add detail including as a description, logo and what versions of Contensis the package is compatible with.

Once created, you can click on the *Developer Packages* button on the left hand side and you'll see your package along with other developer packages that have been created by your organisation. Clicking on one will open a dropdown that will show the versions of the package and when each one was released to the marketplace. You'll also see various action buttons depending on your permissions and whether a package has been installed or needs publishing to live.



Publishing and Installing Developer Packages

Only marketplace developers are able to create, edit and install developer packages. Only marketplace administrators can release developer packages to the Marketplace. If you feel you don't have the appropriate permissions, please speak to you marketplace administrator please speak to one within your organisation.

Handcrafting a Package

It's possible for developers to hand craft a package using XML. They can even define custom settings that could be set against the package when installing it.

Package Structure

A package can contain the following folders and files

- *CMS-content* – Files that will appear in the CMS
- *structured-content* – Any custom structured content xml files should be added to this folder
- *deployed* – Files in this folder will be deployed to the front end website only and will not be editable in the CMS.
- *CMS-contentcontnen.xml* – File that stores Contensis data about the content in the CMS-content

  package.xml – Contains details about the package.

Feedback

If you would like to provide feedback anonymously, please go to https://zengenti.com/en-gb/feedback-surveys/developer-training-feedback.aspx

Resources

Links

- https://zenhub.zengenti.com/Get-Involved/Get-Involved.aspx
- https://zenhub.zengenti.com/Contensis/Marketplace/Marketplace.aspx
- https://zenhub.zengenti.com/Contensis/R82/Development/Razor/ApiControllers.aspx
- https://zenhub.zengenti.com/Contensis/R83/Development/Razor/WebAPI-ContensisForm-Helpers.aspx
- https://zenhub.zengenti.com/Contensis/R82/kb/building-a-website/Data-Listings/data-filter-property-reference.aspx
- https://zenhub.zengenti.com/Contensis/R82/Development/Razor/propertied-razor-views.aspx
- https://zenhub.zengenti.com/Contensis/R82/Development/Razor/node-query.aspx
- https://zenhub.zengenti.com/Contensis/R82/kb/building-a-website/Search/creating-a-search.aspx

Common Content Type Id's and Names

ID	WebApi Name
-1	Folder
0	Web Page
1	HTML
2	Image
3	Template
9	Stylesheet
12	JavaScript
14	Database
15	CSV File
16	Hyperlink
17	MS Word
18	MS Excel
19	MS PowerPoint
21	PDF
22	MS Visio
24	MS Project
28	Executable
29	RAR Archive
30	ZIP Archive
32	XSLT
37	MS Word Template
38	MS PowerPoint SlideShow
39	Windows Media
42	Favourite Icon
43	HTML Upload
44	MS Publisher
46	MS Access

50	FLV Video
52	WMV Video
53	MOV Video
55	MS Excel Template
57	MP4 Video
58	OGG Video
64	MSI File
65	Visual Basic File
70	MS Outlook Mail message
71	Form
73	WebM Video
74	MP3 File
75	MS PowerPoint Template
78	MS Outlook Template File
79	iCalendar

Notes