Sitecore® Platform Essentials for Developers

Student Lab Guide

sitecore®
Own the experience™

CONTENTS

Known Issues

MODULE 1: The Sitecore Experience Platform

In this module, you will learn the Sitecore Experience Platform basic concepts. This includes the interfaces, basic architecture, and how to set up your developer machine to be able to create new Sitecore solutions.

You will also get introduced to the Travel Adventure Company website you will build during this course.

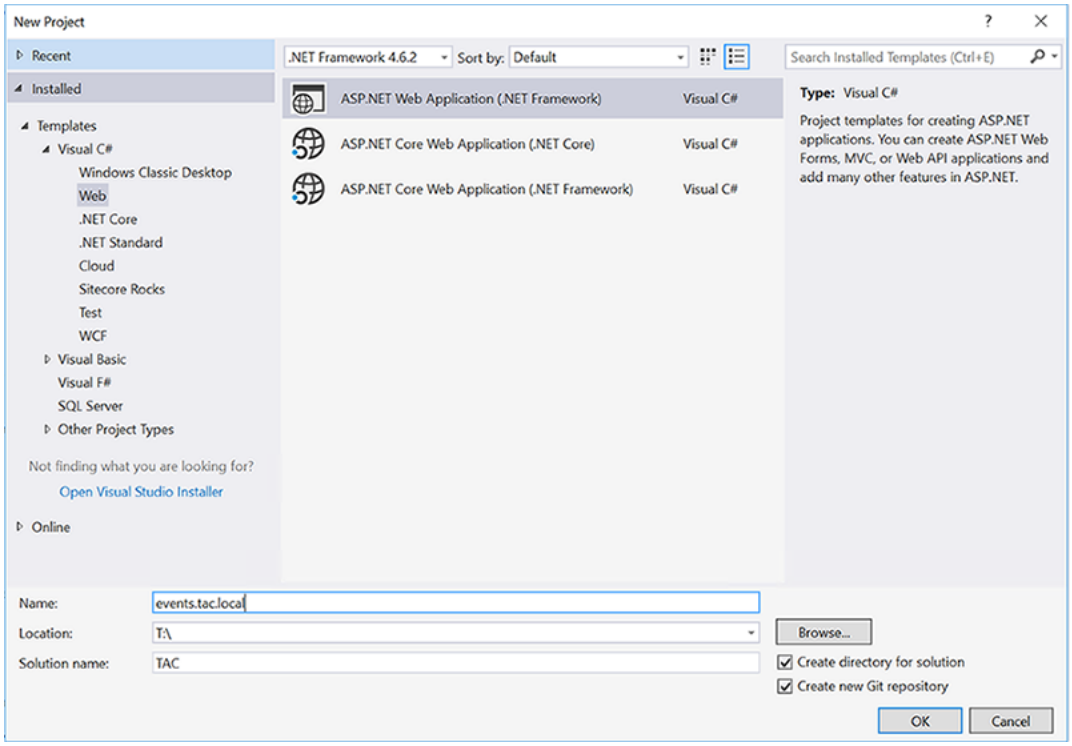
Lab: Setting up a Sitecore Solution

In this lab, you will create and set up a Visual Studio solution to work with Sitecore. You will use this solution to do the exercises during the course.

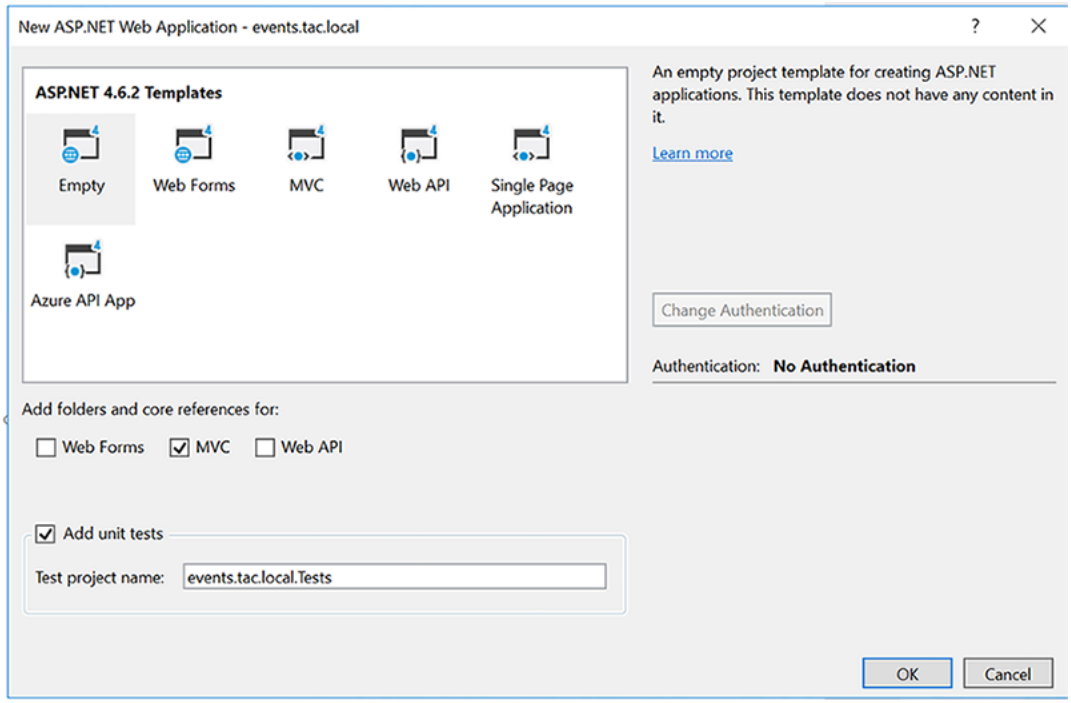
1. To create the project:

Detailed Steps

1. On the Visual Studio menu, choose: **File, New, Project**
2. Select **.NET Framework 4.6.2** from the top drop down.
3. Choose **ASP.NET Web Application.**
4. Name the new project events.tac.local.
5. Select the **Create directory for solution** check box.
6. Name the solution *TAC*.
7. Choose a suitable location in your computer.
8. Optionally, enable **Create new Git repository** if you want to have source control in the solution.



9. In the **New ASP.NET Project** dialog, select **Empty** from the **ASP.NET 4.6.2 Templates** section, select the **MVC** check box and enable **Add unit tests**.

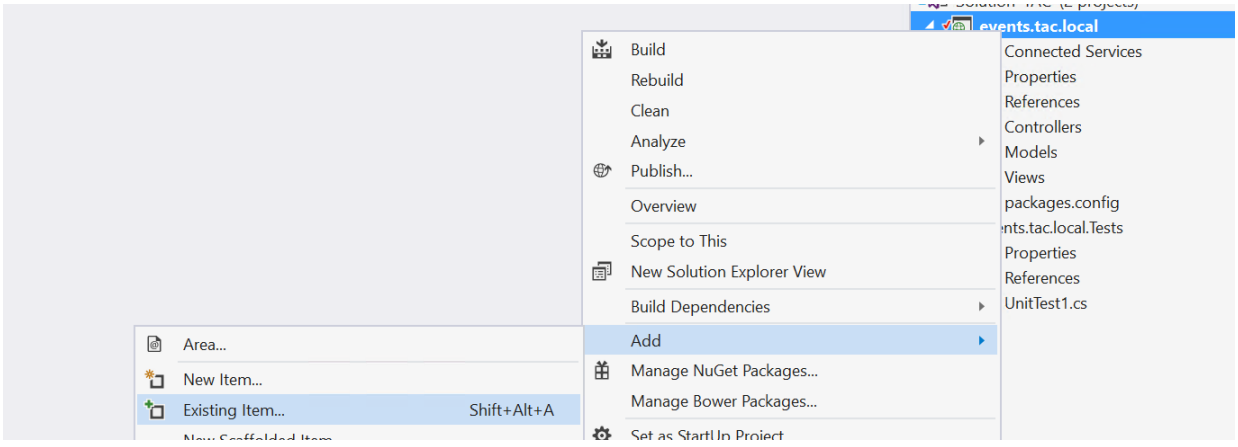


10. Click **OK** to create the project.

2. Setting up the project:

Detailed Steps

1. Delete the **App_Data, App_Start, Web.config** and **Global.asax** files
2. Right-click on the project item and choose **Add > Existing item...**



3. Navigate to the Sitecore installation webroot and choose the **Web.config** file.
4. In the Solution Explorer, find the file Views/web.config file.

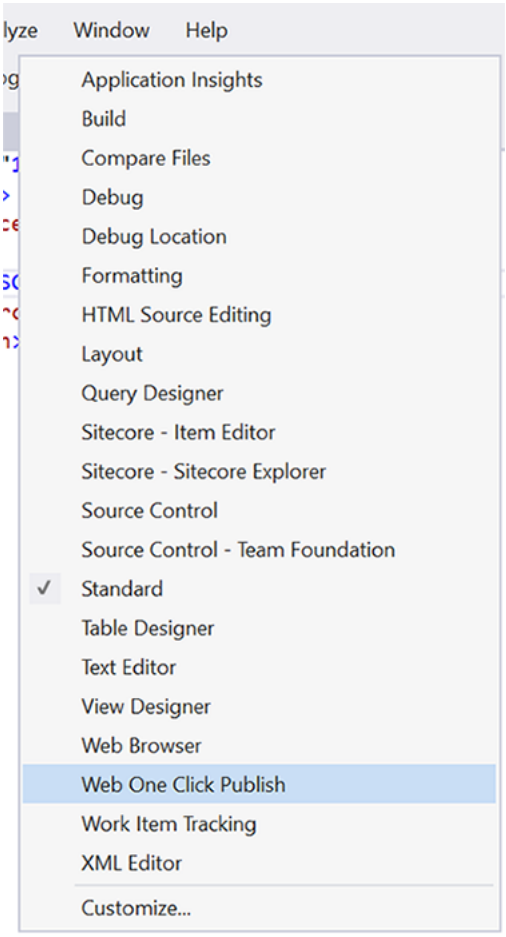
5. Add the following default namespaces:

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFact
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Routing" />
      <add namespace="events.tac.local" />
      <add namespace="Sitecore.Mvc"/>
      <add namespace="Sitecore.Mvc.Presentation"/>
    </namespaces>
  </pages>
</system.web.webPages.razor>
```

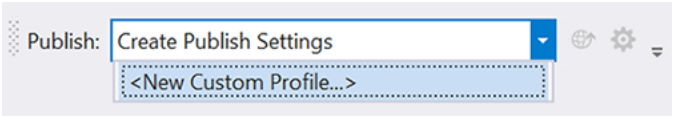
3. Configuring Visual Studio deploy

Detailed Steps

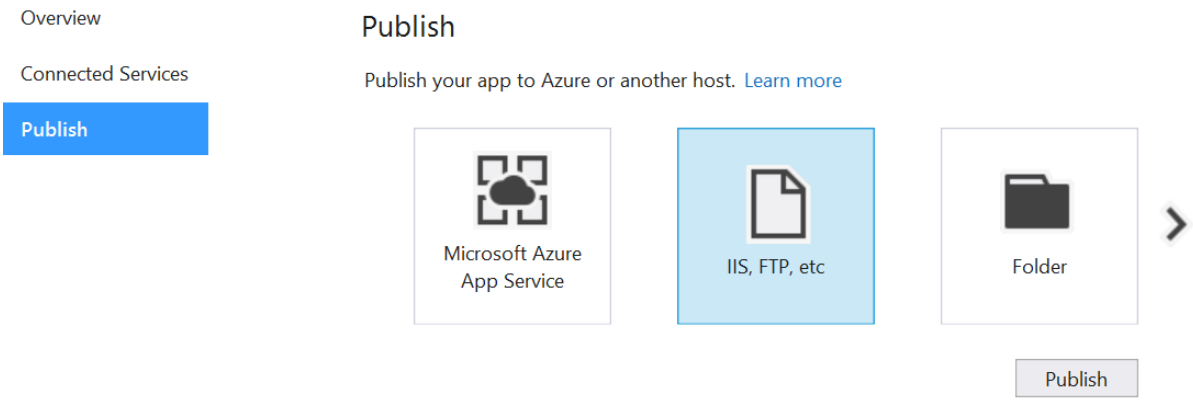
1. Right-click the Visual Studio toolbar and enable the **Web One Click Publish toolbar**.



2. Ensure that you select the **events.tac.local project** in the Solution Explorer.
3. On the Publish dropdown menu, select **<New Custom Profile...>**.



4. Choose **IIS, FTP, etc..**



5. Click **Publish**
6. In the Publish method, select **File System**.
7. Add the path to the webroot in the **Target location** and click **Next**.
8. On the Configuration dropdown menu, choose **Debug** and click **Save**.
9. Open the **CustomProfile.pubxml** (which is in the Properties/PublishProfiles folder of the project).
10. Just before the **PropertyGroup** closing tag, add the following line:

```
  <DeleteExistingFiles>False</DeleteExistingFiles>
  <ExcludeFilesFromDeployment>bin\Sitecore.Kernel.dll;bin\Sitecore.Mvc.dll;bin\Sitecore.Mvc.Analytics.dll;Web.config</ExcludeFilesFromDeployment>
</PropertyGroup>
```

```
      <DeleteExistingFiles>Files</DeleteExistingFiles>
               <ExcludeFilesFromDeployment>
                     bin\Site-
core.Kernel.dll;bin\Sitecore.Mvc.dll;bin\Sitecore.Mvc.Analytics.dll;Web.config
              </ExcludeFilesFromDeployment>
</PropertyGroup>
```

11. Save and close the file.
12. Every time you need to compile and deploy your solution, you can use the One-Click icon:

4. Configuring NuGet

Detailed Steps

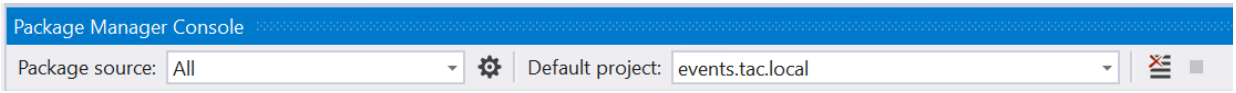
1. In the **Student Resources** folder, locate the **NugetReferences.zip file**.

2. Create a folder called *Sitecore NuGet* in your solution's **TAC** folder.
3. Unzip the **NuGetReferences.zip file** into the **Sitecore NuGet** folder that you just created.
4. In Visual Studio, use the **Manage NuGet Packages** feature to add a new **Package Source**, which references your **Sitecore NuGet** folder. Call it *Sitecore NuGet*.

5. Adding Sitecore NuGet packages:

Detailed Steps

1. Open the **Package Manager Console** using either the **Quick Launch** or the **Package Manager Console** (Click **View, Other Windows, Package Manager Console**).
2. On the Package Source dropdown menu, ensure **All** is selected and choose **events.tac.local** as the default project.



3. In the console, type the following commands to install the relevant Sitecore packages:

```
Install-Package Sitecore.Kernel.NoReferences -Version 9.0.0
Install-Package Sitecore.Mvc.NoReferences -Version 9.0.0
Install-Package Sitecore.Mvc.Analytics.NoReferences -Version 9.0.0
```

6. Configuring the test project:

In this course, you will use **xUnit** so it needs to be added to the test project's references.

Detailed Steps

1. In the events.tac.local.Tests project, remove either the reference to *Microsoft.VisualStudio.TestPlatform.TestFramework*, or *Microsoft.VisualStudio.QualityTools*.UnitTestFramework, depending on which version of Visual Studio that you are using.
2. Open the **Package Manager Console**. Ensure the selected default project is **events.tac.local.Tests**
3. Install the following modules:

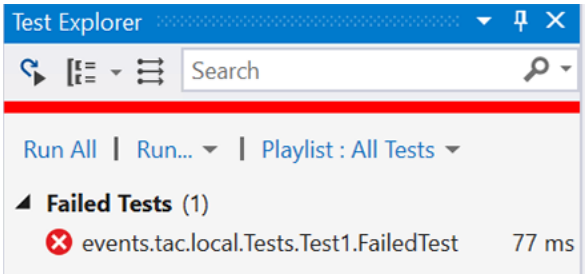
```
Install-Package xunit
Install-Package xunit.runner.visualstudio
```

4. **Delete** the file **UnitTest1.cs** from the test project.
5. Create a new class *Test1.cs*

6. Add the following code:

```
using Xunit;

namespace events.tac.local.Tests
{
    public class Test1
    {
        [Fact]
        public void FailedTest()
        {
            Assert.True(false);
        }
    }
}
```

7. Open the **Test Explorer** window (Click **Test, Windows,Test Explorer**)
8. Click **Run All.** You should not see any compilation errors and that the test failed, as shown in the image below. This is intended to ensure that you have configured this **events.tac.local.Tests** project correctly because it will be used later.



STOP HERE

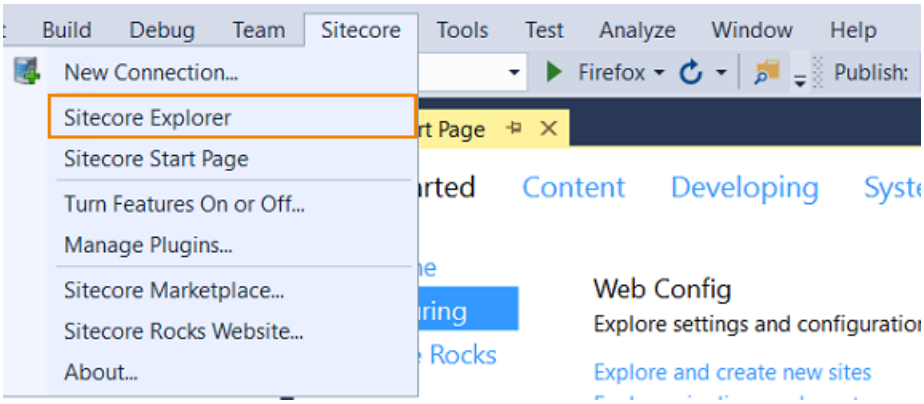
Lab: Setting up Sitecore Rocks

In this lab, you will configure a connection to your Sitecore instance in Visual Studio using Sitecore Rocks. You will then associate this connection with your Visual Studio project.

1. To create a Sitecore Rocks connection:

Detailed Steps

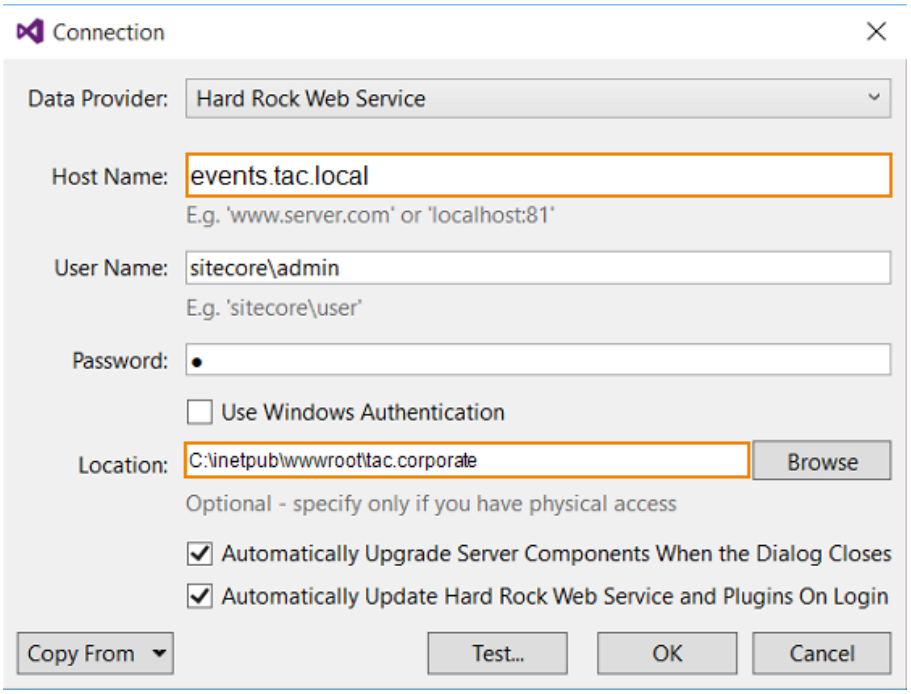
1. Open **Visual Studio.**
2. Verify that the **Sitecore Explorer** is visible. If it is not visible, you can open it by clicking the Sitecore menu in the toolbar and choosing **Sitecore Explorer.**



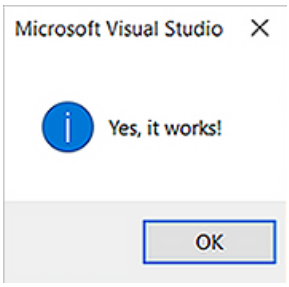
> The Sitecore Explorer can float or be docked like any other Visual Studio window.

3. In the **Sitecore Explorer**, right-click the **Connections** folder and choose **New Connection...**
   a. In the **Connection** dialog, enter:
      - Host Name: *events.tac.local*
      - Location: *C:\inetpub\wwwroot\tac.corporate*

> **NOTE:** The location is set to the path of the webroot; found by right-clicking on the site in IIS, and selecting **Explore.**
> This optional setting allows Sitecore Rocks to install the web service it requires for communicating with the Sitecore instance. If you leave the location blank, then you will need to install those files manually.

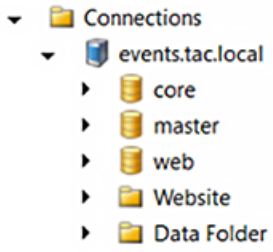


b. Click **Test**... to see if your connection works.



> **NOTE:** Sitecore Rocks may request updated server components when you test or create the connection. If you are prompted to update, allow it to update any necessary components.

4. Click **OK** to close the dialog. Your connection has been created.



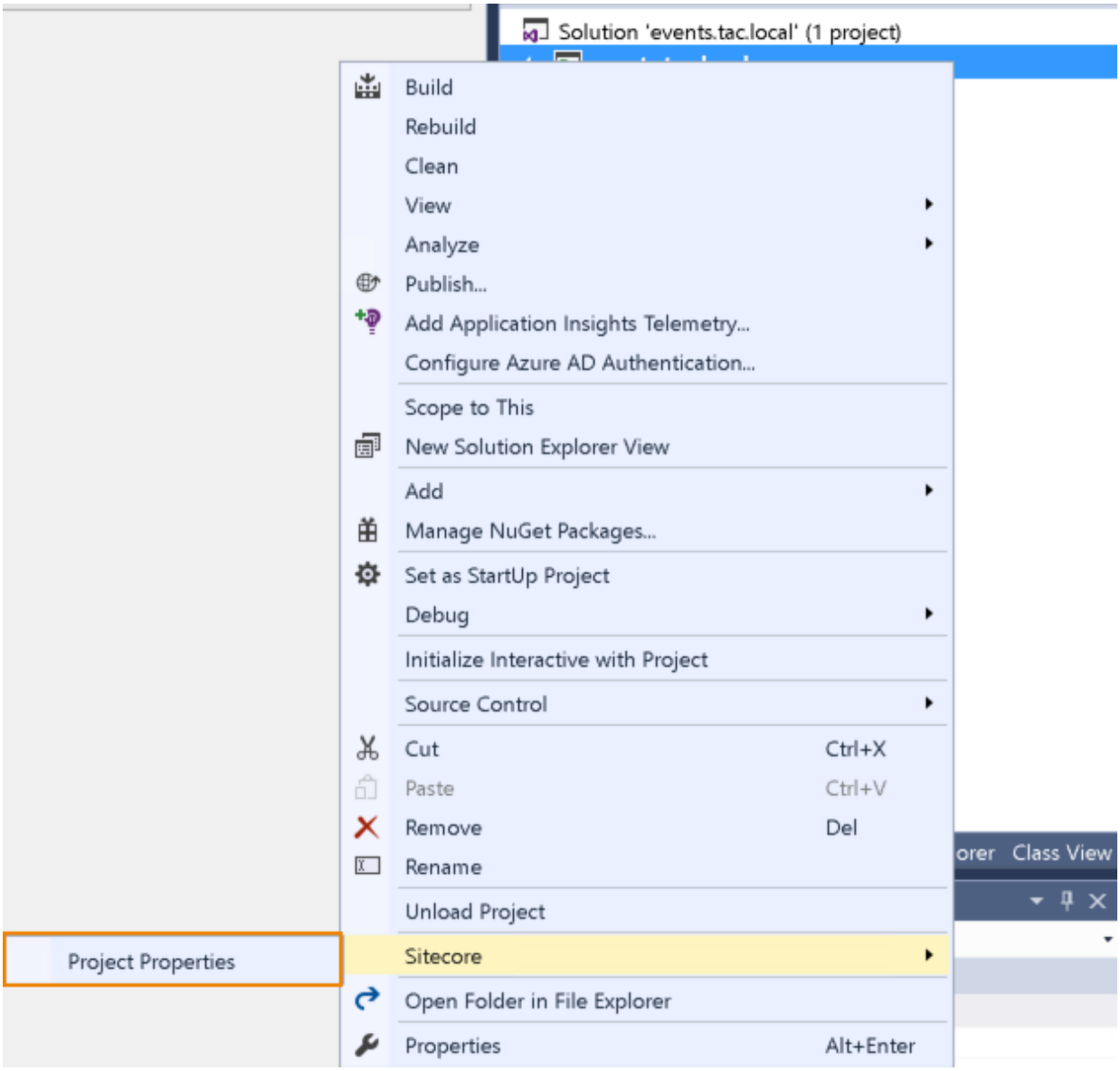
2. To connect Sitecore Rocks to your Visual Studio project:

Now that you have configured a Sitecore Rocks connection, you can associate this connection with your Visual Studio Project. Sitecore Rocks adds Sitecore templates and other functionality to Visual Studio to provide a deep integration between your project and Sitecore instance.

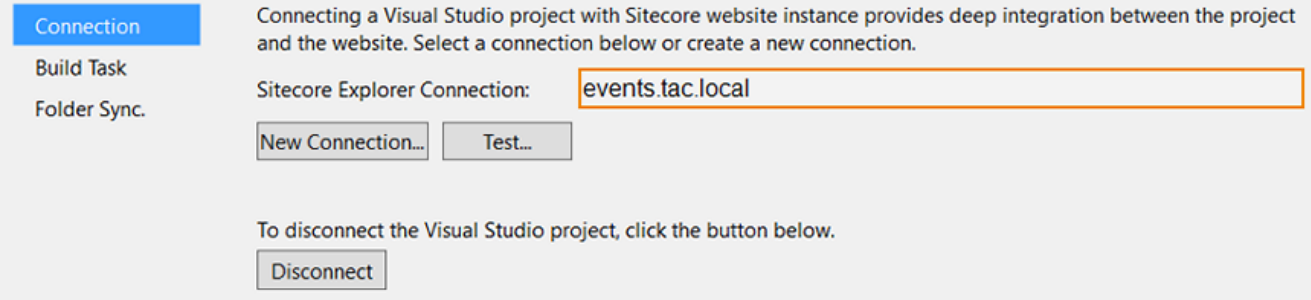
Detailed Steps

1. In **Visual Studio Solution Explorer,** right-click the **events.tac.local** project to display the context menu.

2. Choose **Sitecore, Project Properties.**



3. You configure your connection between a Sitecore instance and Visual Studio on the **Project Properties** tab. In the Sitecore Explorer Connection, choose **events.tac.local**.



On the Project Properties tab, you can also create and test Sitecore Rocks connections like you did in the previous set of steps. You can also disconnect your Visual Studio project from your Sitecore instance here.

NOTE: You now have two Explorers open, the Sitecore Explorer and Solution Explorer.

- Use the **Sitecore Explorer** to create and edit Sitecore items and any files in the Data and Website folders.
- Use the **Solution Explorer** to create code files (using Sitecore Templates when needed) and create and manage configuration patches.

STOP HERE

MODULE 2: Creating the Site's Structure

In this module, you will define the templates that you will use on your site that will form the basis of Content Items. You will also make use of template inheritance, Standard Values, Insert Options, and default fied values.

Lab: Creating Templates for the Events Site

In this lab, you will use Sitecore Rocks to create the templates that you need to build your site's content. You will also configure inheritance and assign icons to your templates.

> Sitecore Rocks offers a convenient shortcut called Commandy (Ctrl + Shift + Space) to open common commands. Try to use Commandy during this lab by selecting an item in the Sitecore Explorer and invoking the command.

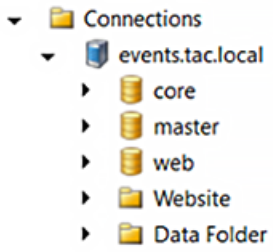
1. Scoping Sitecore Rocks connections:

The Scope feature in Sitecore Rocks let's you hide sections of the content tree in the Sitecore Explorer and focus on where you are working. Often you are involved in many Sitecore projects and would like to hide other Sitecore instances, or perhaps you want to only display the Master database's items for your current project.

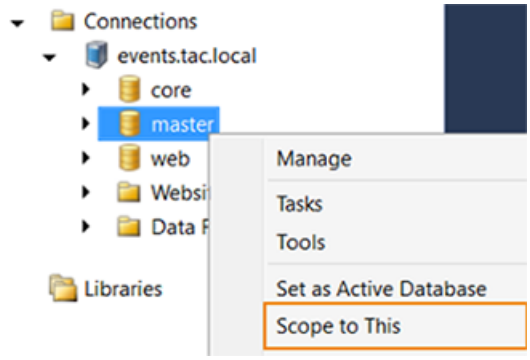
The following steps will show you how to scope to the Master database. This not only allows you to focus on your work area but prevents you from creating and editing items in other databases unintentionally.

Detailed Steps

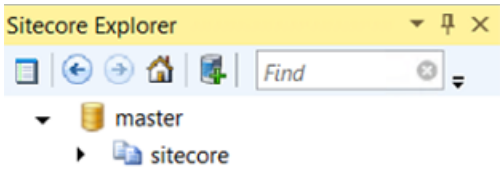
1. Open Visual Studio and if it's not already open, and show the **Sitecore Explorer** from the menu: *Sitecore* > *Sitecore Explorer*. If you do not see the Sitecore menu in Visual Studio, install Sitecore Rocks.



2. Right-click the Master database and choose *Scope to This* to hide the other databases.



Below you see the result of the end result after you have performed the scoping action.



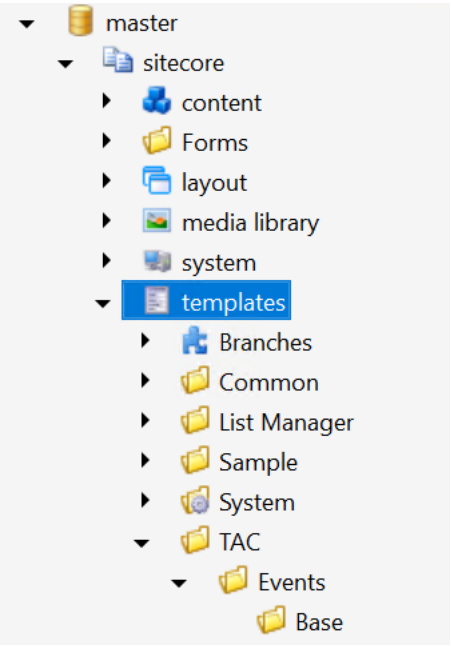
2. Adding the template folder structure

You will create templates in the /sitecore/templates section of the tree. In order to assist with maintenance and avoid conflicts you will create your own folder to store those templates. You might have to create tens or even hundreds of templates in a Sitecore solution, you will usually create further folders to classify them.

Detailed Steps

1. In the **Sitecore Explorer** navigate to the **/sitecore/templates** node.
2. Right-click and select **New Template Folder**.
3. Name the folder *TAC* .
4. Create another Template folder inside TAC named *Events*

5. Create another Template folder inside Events named *Base*.

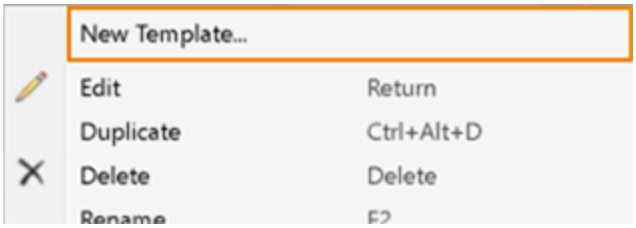


3. Creating a base template:

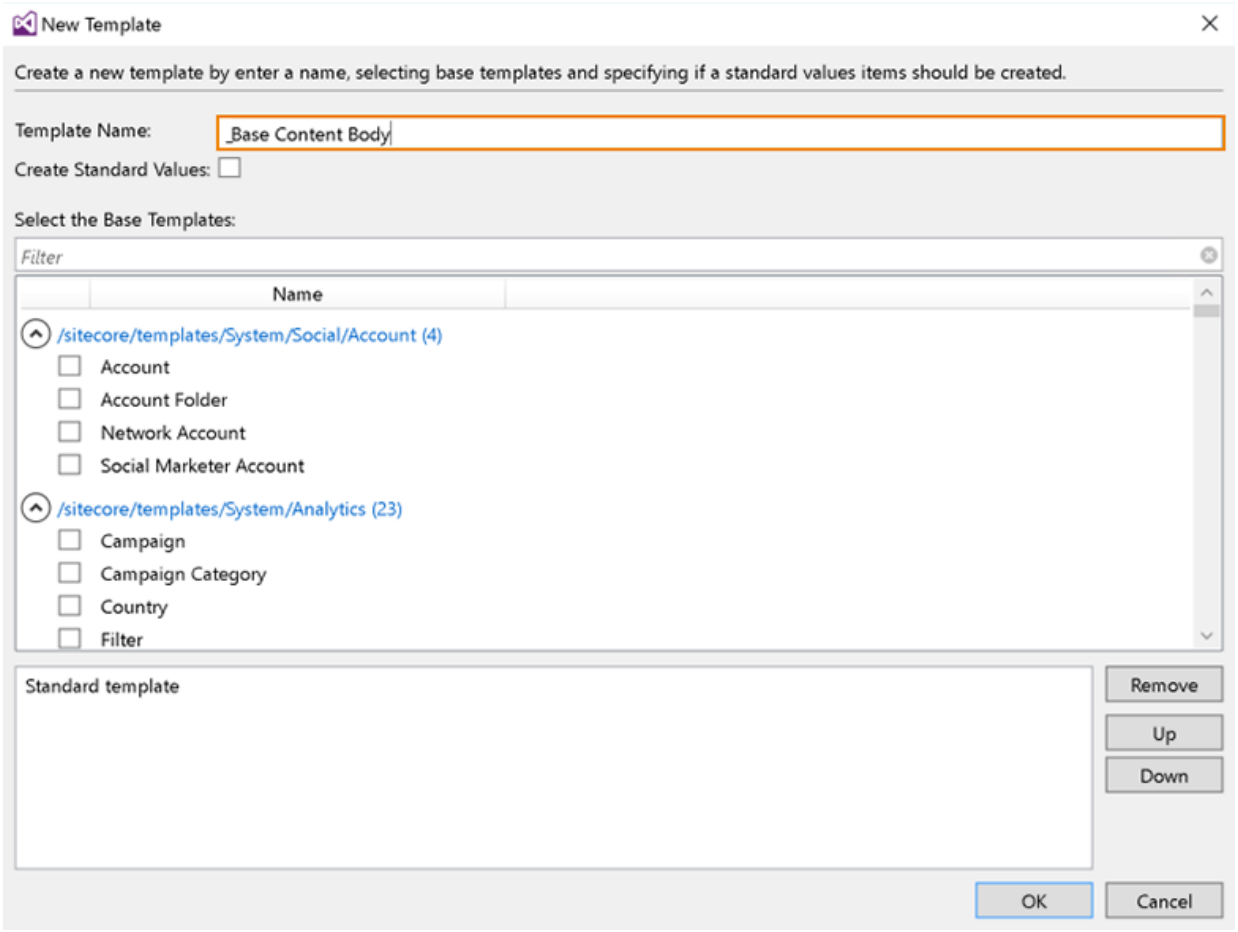
You will start by creating the base templates that allow you to re-use the same field definitions. When creating a template you are defining the fields that will be used to store content. Those fields are grouped into field sections.

Detailed Steps

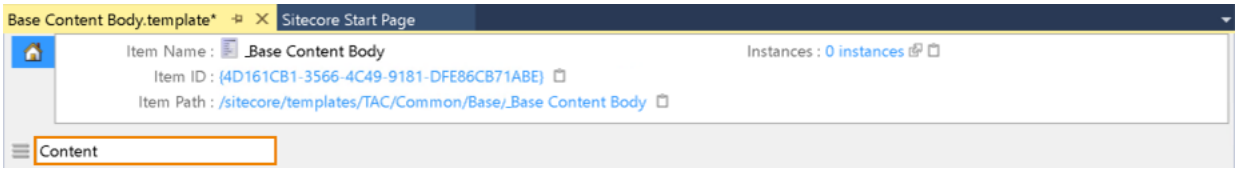
1. In the Sitecore Explorer, in Visual Studio, navigate to: *sitecore/templates/TAC/Events/Base*
2. Create a new template by right-clicking the **Base** folder and choosing **New Template...**



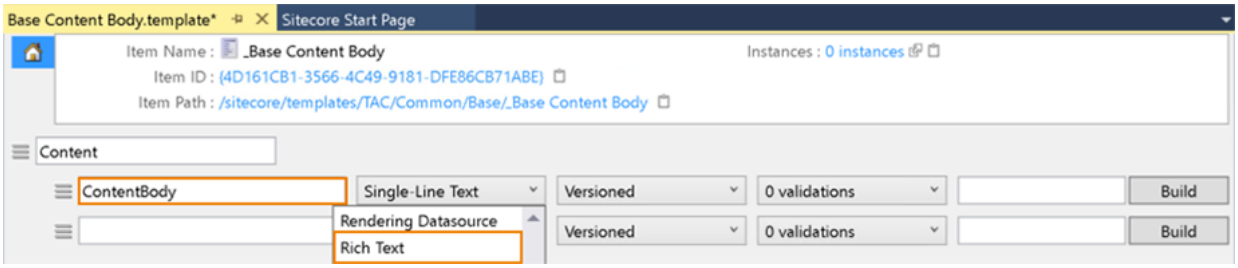
3. Name the new template *_Base Content Body* and click **OK**.



4. Create a new section and name it *Content*.



5. Create a new field and name it *ContentBody* and select the **Rich Text** field type.



6. **Save** your new template by using **Ctrl+S**.
7. Repeat the previous steps to create the following base templates:

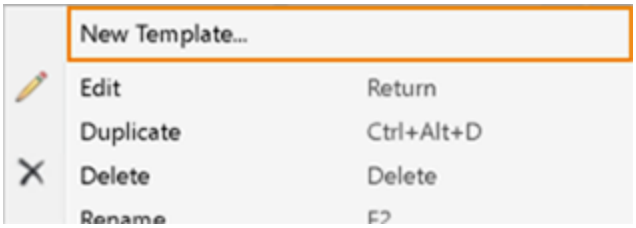
Template	Section	Field	Field type
_Base Content	Content	ContentHeading	Single-line Text
		ContentIntro	Multi-line Text
_Base Meta Information	Meta	MetaDescription	Single-line Text
		MetaTitle	Single-line Text
	Robots	RobotsFollow	Check box
		RobotsIndex	Check box
_Base Heading Decoration	Decoration	DecorationBanner	Image
_Base Navigation	Navigation	ExcludeFromNavigation	Check box

4. Creating a new template that inherits from existing base templates:

You will now create other templates that inherit from the base templates you just defined. This will allow you to reuse those field definitions.

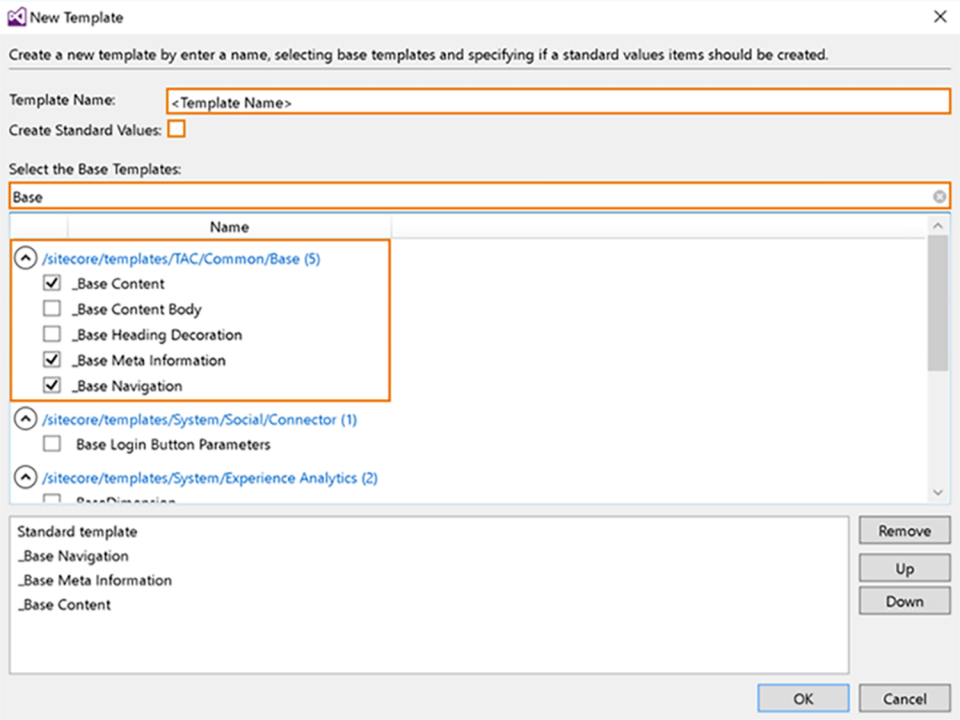
Detailed Steps

1. Right-click the **Events** template folder and choose *New Template...*



2. The new template's name is: *Home Page*.

3. Use the Base Templates search filter and enter: *Base*. Select the following base templates:
    - _Base Content
    - _Base Meta Information
    - _Base Navigation



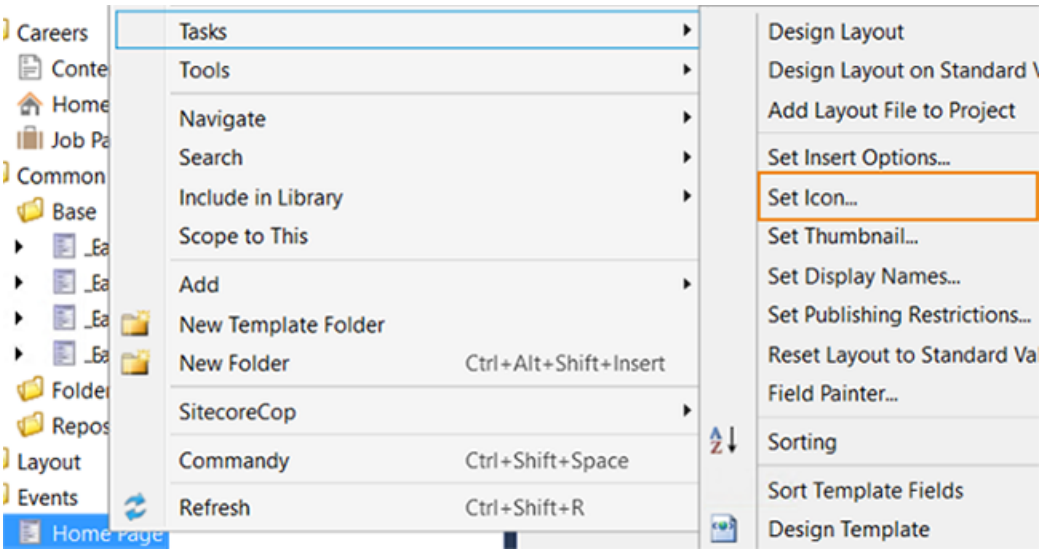
4. Click **OK.**

> **Recommended practice:** Use template inheritance to reduce field redundancy.

5. Setting a template icon:

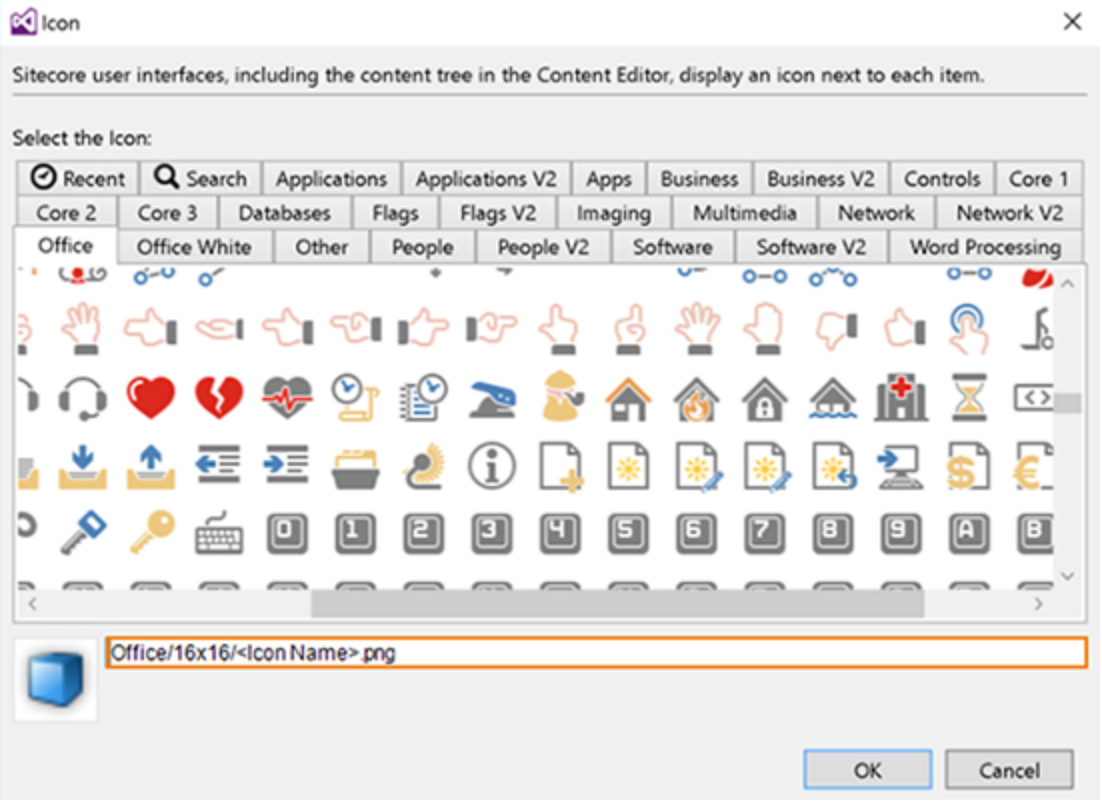
Detailed Steps

1. Right-click the new template and choose *Tasks > Set Icon...*



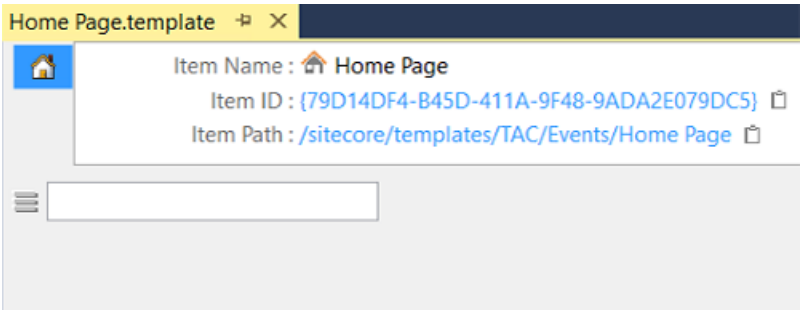
Make good use of icons to speed user recognition when navigating your site's content tree.

2. Select a relevant icon for your new template. For the Home page template, you can use the home icon (*Office/16x16/home.png*) from the Office repository. Click **OK** when you are done.



When you know the path to the icon that you want to use, you can directly insert the path in the bottom text box of the Select Icon window. You can use the Search tab to search for icons.

3. You have now created the new template and if needed you can start adding field sections and fields to it. In this case this template does not require any further fields, as it is already inheriting from the parent templates all the fields it needs.



> **NOTE:** Inherited fields are not displayed in the Template Editor. All you see is the Field Section text box on the Home Page template. The inherited fields will appear when you create content from this template.

4. **Save** your changes.

6. Creating the remaining Events templates:

Detailed Steps

1. Repeat the steps for creating templates with inheritance to create the remaining templates in the Events folder, according to the following table. Don't forget to set the icons.

Template Name	Base Templates	Icon
Content Page	• _Base Content • _Base Content Body • _Base Heading Decoration • _Base Meta Information • _Base Navigation	Office/16x16/document_tex-t.png
Events Section	• _Base Content • _Base Heading Decoration • _Base Meta Information • _Base Navigation	Office/16x16/window_time.png

Template Name	Base Templates	Icon
Events List	• _Base Content • _Base Heading Decoration • _Base Meta Information • _Base Navigation	Office/16x16/history2.png
Event Details	• _Base Content • _Base Content Body • _Base Heading Decoration • _Base Meta Information • _Base Navigation	Office/16x16/pin.png

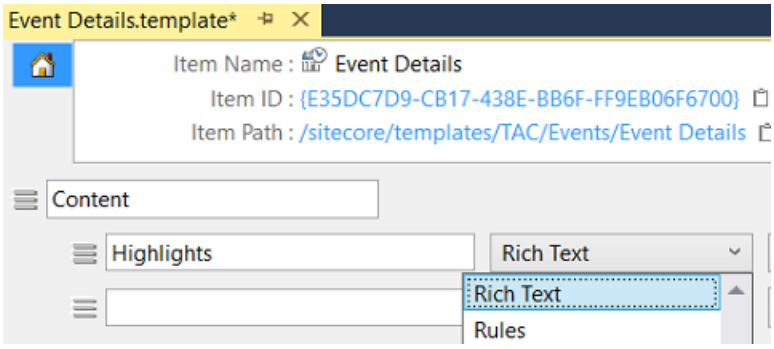
7. Add fields to the Event Details template:

Detailed Steps

1. Open the **Event Details** template.
2. On the Editor tab, add a **field section**: *Content*

> **NOTE:** The _Base Content and the _Base Content Body templates both already contain a Content field section. All field sections with the same name will merge into one section in the final template.

3. In the Content field section, add a **Rich Text** field and name it: *Highlights*.

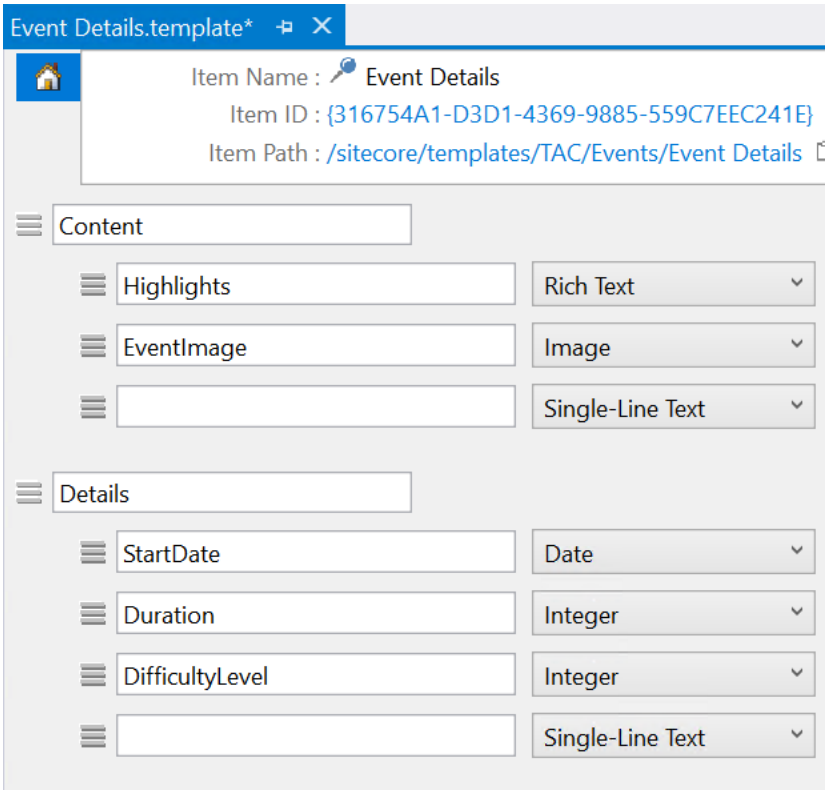


4. Add an **Image** field. Name: *EventImage.*
5. Add another **field section** : *Details.*

> Use field sections to group related sets of information. This is an organizational structure for users.

6. Add a **Date** field. Name: *StartDate.*

7. Add an **Integer** field. Name: *Duration.*
8. Add another **Integer** field. Name: *DifficultyLevel.*
9. **Save** your changes.



Here is the completed **Event Details** data template.

STOP HERE

Lab: Adding Content in Sitecore Rocks

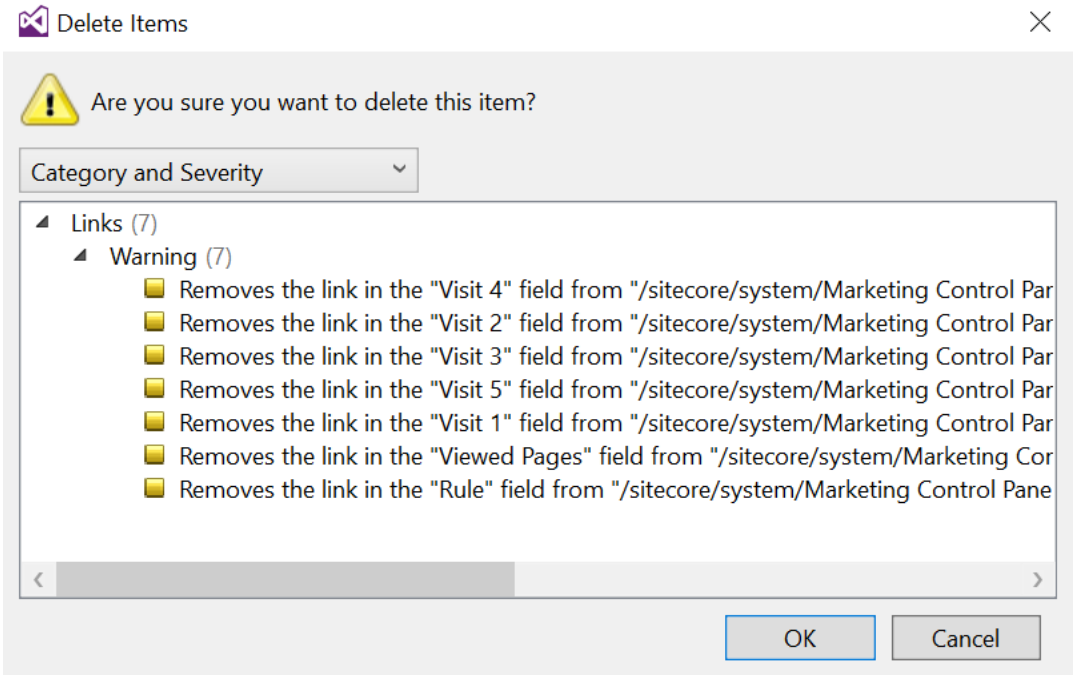
In this lab, you will add the basic site structure for the Events section and Events categories.

1. Replacing the Home item

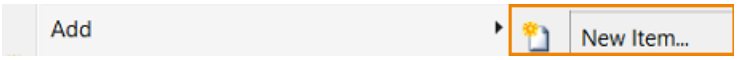
Detailed Steps

1. In the Sitecore Explorer use the events.tac.local connection to locate and select the */sitecore/content/Home item*.
2. Press the **<delete> key**.

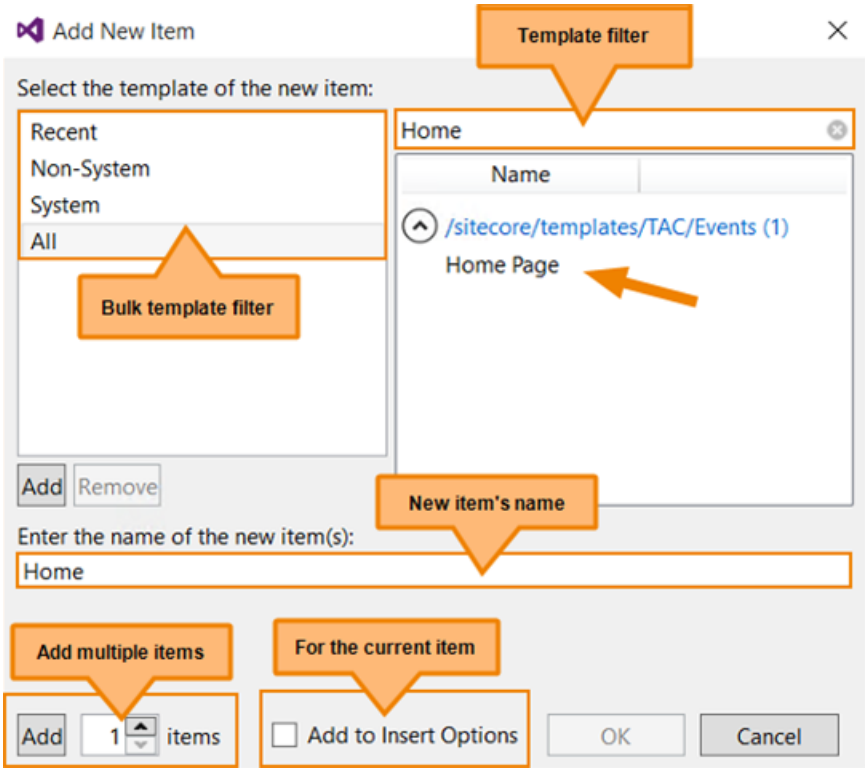
3. If the broken links report appears, click **OK**.



4. Right-click **Content**, choose: **Add, New Item.**



5. Find the **Home Page** template by typing "*Home*" into the filter text box, select **Home Page** from the **TAC/Events** folder. Name the new item: *Home*



6. Click **OK**.

2. Editing items in Sitecore Rocks

Now that the item has been added, you can start populating the new item with content.

Detailed Steps

1. Edit the following fields

Field Name	Value
ContentHeading	Events
ContentIntro	Welcome to the Events site
MetaTitle	The Adventure Company - Events

2. **Save** your changes.

3. Create the site folder structure

Now that you have a new Home item you will use the Add New Item command to create an Events section.

Detailed Steps

1. Right-click the **Events/Home** item and choose: *Add, New Item*
2. In the **Add New Item** dialog, filter for "*events*" and select the **Events Section** template from the list. Name the new item: *Events*

3. Click **OK**.

STOP HERE

Lab: Creating Standard Values

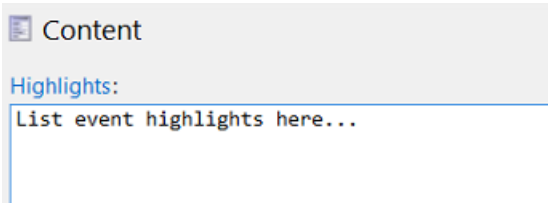
1. Setting default field values:

In this lab, you will create standard values and assign the correct insert options. Standard values are essential for defining default values and for setting default configurations like insert options and presentation.

Detailed Steps

1. In Sitecore Explorer, select the **Event Details** template.
2. Right-click **Event Details**, select: *Create Standard Values.*

3. In the **Highlights** field enter: *List event highlights here...*



4. **Save** your changes and then **close** the Editor tab.

2. Setting default field values in base templates:

You can also set default values further up the inheritance chain. This is really useful as those default values are applied to any inheriting template, although they can also be overridden. Here you will add a token so the ContentHeading field is automatically replaced with the name of the item created. You will then repeat the steps to also apply the same initial value to the MetaTitle field.

Detailed Steps

1. In Sitecore Explorer, select the **_Base Content** template.
2. Right-click **_Base Content**, select: *Create Standard Values.*

3. In the **ContentHeading** field enter: *$name*.
4. **Save** your changes and then **close** the Editor tab.
5. Repeat the steps above to set *$name* as the standard value for the **MetaTitle** field in the **_Base Meta Information** template.

3. Testing the effect of default values:

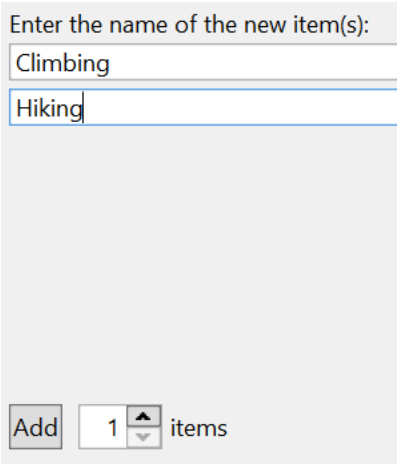
Create two new items under /sitecore/content/Home/Events named Climbing and Hiking, based on the Events List template.

Detailed Steps

1. To create multiple items at one time use the **Add > New Item** command.
   Right-click the **/sitecore/content/Home/Events/** item and choose: *Add > New Item* (or use the shortcut Alt+Insert)
2. In the **Add New Item** dialog, filter for "event", select the Events List template from the list.
3. Click the Add button in the lower left corner.

> The Add command always adds the number you have entered to the existing item to be added. If you want three new items you would enter 2 in the Add counter input.

4. Name the items *Climbing* and *Hiking* and press **OK**.



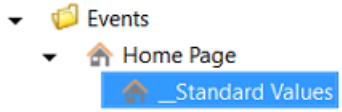
5. Click the tab labeled **2 items.item** to close the bulk editing window, and open the **Climbing** item by double-clicking it. Notice how the ContentHeading field is already populated. This is because the $name token was added to the _Base Content template standard values. The same is true of the inherited MetaTitle field.

4. Setting insert options:

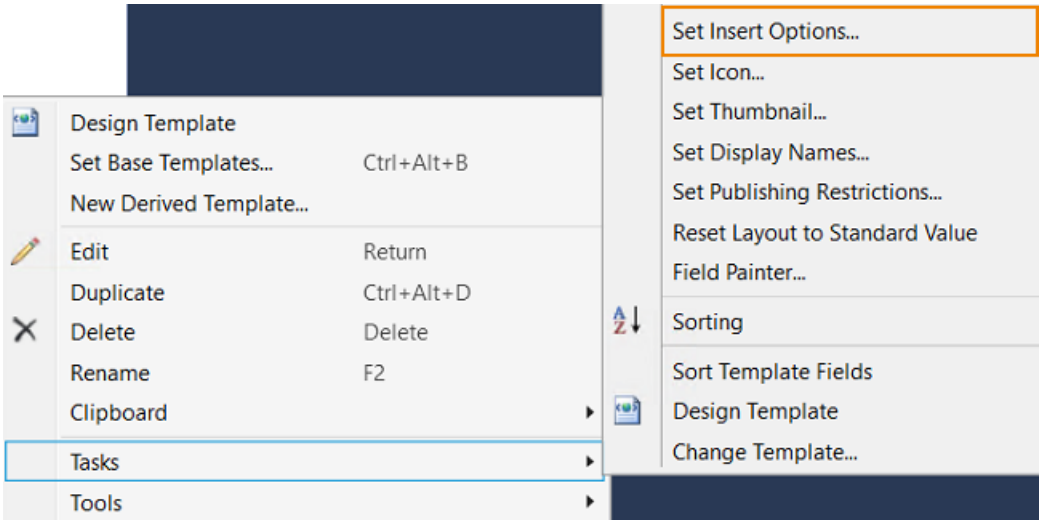
To enable authors to create content, you must define insert options. Insert options are the list of templates that authors can use to create items under the selected content item. The recommended practice is to set those in the Standard Values.

Detailed Steps

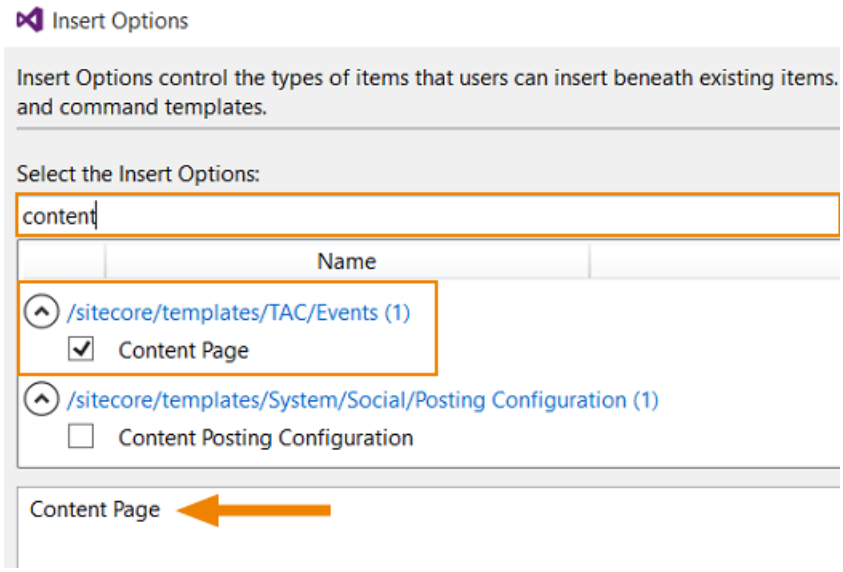
1. In Sitecore Explorer, select the **Home Page** template.
2. Right-click **Home Page** under the Events folder, select: *Create Standard Values.*



3. Right-click the **Home Page Standard Values** item and, from the context menu, choose: *Tasks* > ***Set Insert Options***.



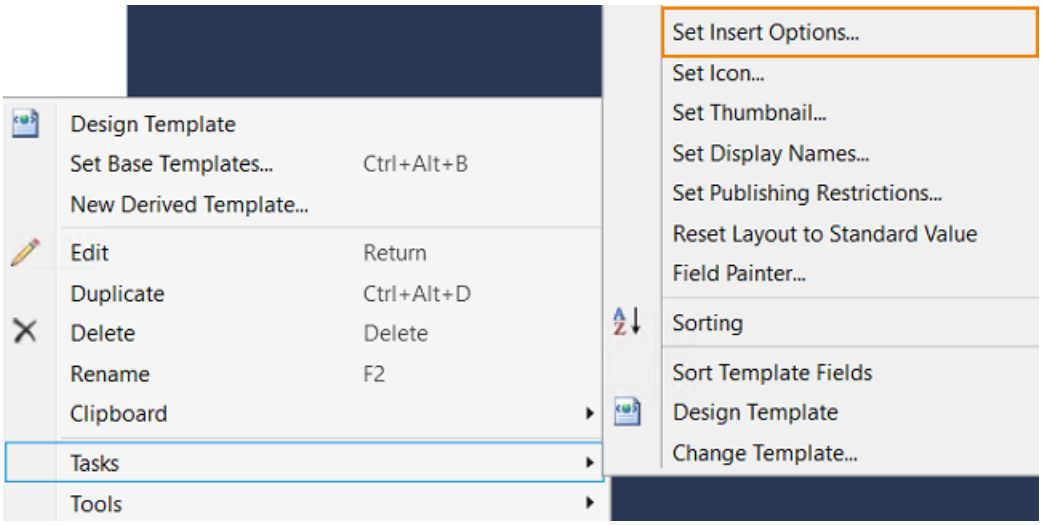
4. In the **Insert Options** dialog, type *content* in the Search box and select *Content Page* from the filtered list. Once you select the Content Page, it will appear in the list box of selected templates at the bottom-left of the dialog.



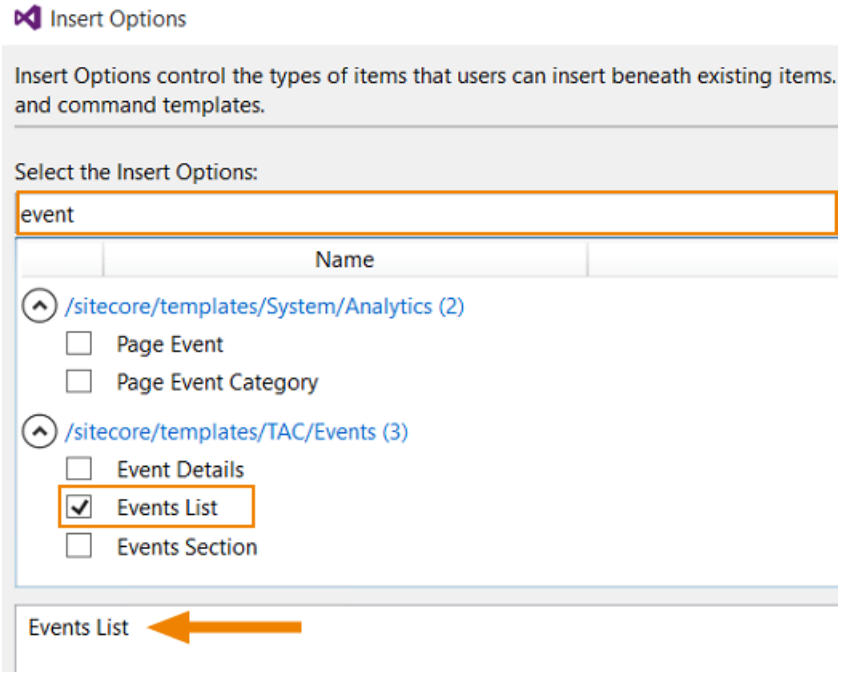
5. Click **OK**.
6. In Sitecore Explorer, select the **Events Section** template.

7. Right-click **Events Section** and select: *Create Standard Values*.

8. Right-click the **Events Section Standard Values** item and from the context menu choose: *Tasks* > ***Set Insert Options.***



9. In the **Insert Options** dialog, type *event* in the Search box and select *Events List* from the filtered list. Once you select the Events List, it will appear in the list box of selected templates at the bottom-left of the dialog.



10. Click **OK**.
11. Add a **Standard Values** item to the **Events List** data template and set its insert options to: *Event Details*.
12. Add a **Standard Values** item to the **Content Page** data template and set its insert options to: *Content Page*.

> **NOTE:** By allowing a Content Page to have child items of the same type, you are enabling authors to create a free structure within the site.

STOP HERE

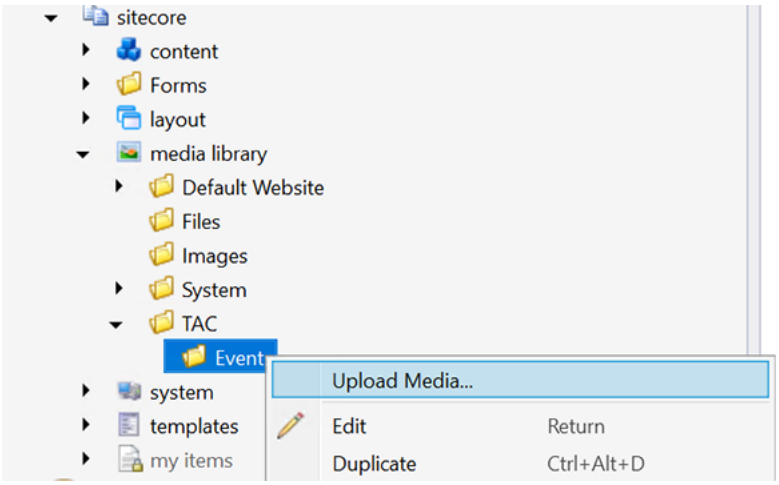
Lab: Adding Other Types of Content

In this lab, you will import images to the Media Library and then create additional content for testing the site later on.

1. Uploading media

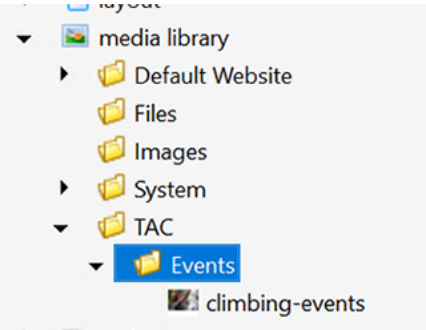
Detailed Steps

1. Right-click the */sitecore/media library* item and choose **New Media Folder** in the **Sitecore Explorer**.
2. Name the folder TAC.
3. Inside the TAC folder, create another Media Folder named Events.
4. Right-click the **Events** media folder and choose: *Upload Media...*



5. In the **Student Resources** folder that you downloaded for this course, navigate to the Images folder and select: *climbing-events.jpg.*

6. Click **Open**. Your Events media folder should look like this:



If you are running Visual Studio as a **regular user** you should be able to drag and drop files directly from Windows Explorer onto the Media Library on the Sitecore Explorer.

2. Assigning media to an image field

Detailed Steps

1. Open the */sitecore/content/Home/Events/Climbing* item for editing.
2. Click the **DecorationBanner** field label. Select the **Browse for Image...** option.
3. In the **Find** text box, type *climbing* and press **enter**.
4. The Climbing-Events media item should be selected. Otherwise expand the tree to locate it. Press **OK** to close the dialog.

   The field menu also has an option for uploading an image from the file system to the root folder of the Media Library and then insert it in the field. Then you can move the media item to the correct folder. The image field will still point to the right image because it stores the ID, not the path, of the media item.

   TEXT CAPTION: Alternatively, you can also drag and drop media items from the Sitecore Explorer to an image field when editing an item.

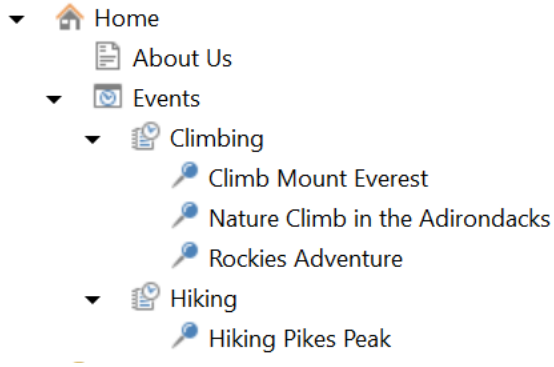
3. Adding content

Detailed Steps

1. Using the Event Details template, create the following items under the */site-core/content/Home/Events/Climbing item*. You can use the Add > New Item command's bulk insert feature, or the Add menu's Event Details **insert option**.

Item Name
Climb Mount Everest
Nature Climb in the Adirondacks
Rockies Adventure

2. Create another item named *Hiking Pikes Peak* under Hiking.
3. Create a Content Page item under Home and name it *About Us*. Your content tree should look like this:



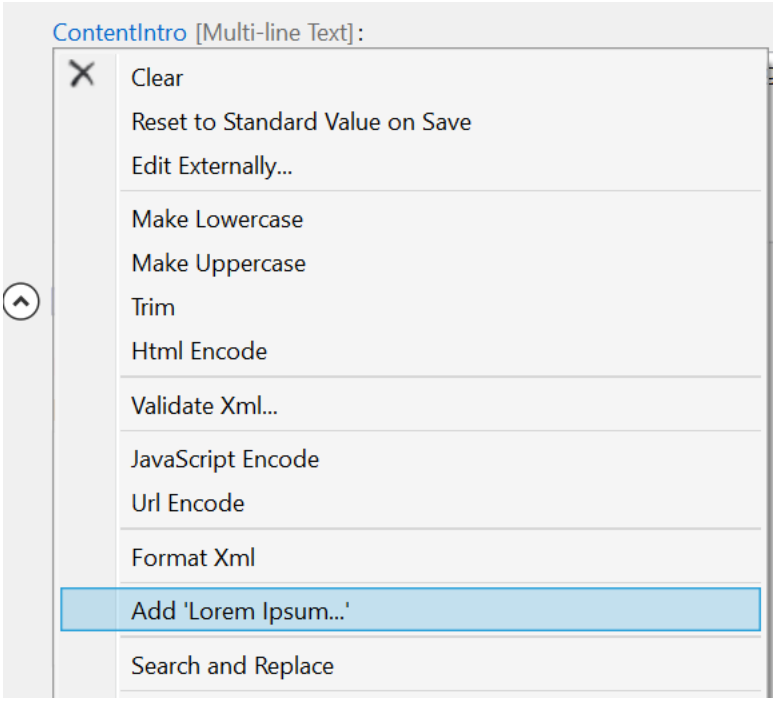
4. Populating fields with test content

Now you will add some dummy content in the fields for those items that you have created. You can use the Lorem Ipsum functionality of Sitecore Rocks.

Detailed Steps

1. Open the **About Us** item.

2. Click the **Content Intro** field label and select **Add Lorem Ipsum...**



   If you want longer content, repeat the previous step as necessary.
3. **Save** the changes.
4. Use the images from the **Student Resources** folder and **Lorem Ipsum** to populate data in all the **Event Details** items.

STOP HERE

MODULE 3: Creating the Site's Presentation

In this module, you will learn about the presentation layer of Sitecore. You will start by learning about layouts and presentation details, the two main building blocks for presentation, and how to render the content from Sitecore as part of the HTML. You will then learn the importance of splitting the page into smaller parts called components. There are different types of components available and you will learn how to use them and add them to the output either statically or dynamically. Finally you will learn about the necessary configuration to allow authors to modify the design of the page using the Experience Editor.

Lab: Creating a Layout and Setting Presentation Details

In this lab, you will create the layout that you will eventually use for the whole site. You will use the **Event Detail HTML template** as the basis for the layout. This is the most complex template because of the stylesheets and JavaScript that is used.

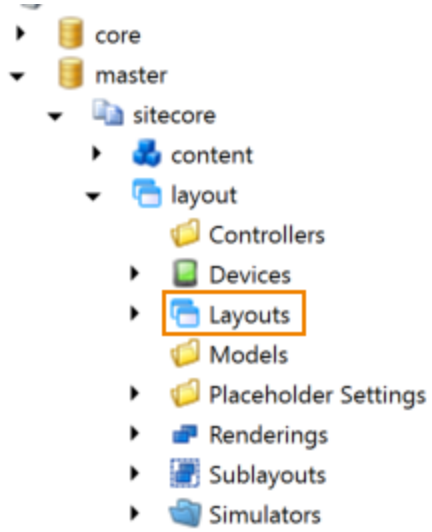
First, though, you will add folders to your Visual Studio project to store the layout code file. Finally, to be able to use the layout to build a response when requesting items. You will assign it to the standard values of all the templates that define items representing pages of the website.

1. Creating the folder structure in Sitecore and Visual Studio

To store the layout code file in the right location of the file system, you will create some new folders in your Visual Studio project. All presentation related files in an MVC project are stored under the Views folder. We recommend that you store your files within a subfolder to avoid conflicts. In this case, you will create the following folder structure in your Visual Studio project: *Views/TAC/Events/Layouts/*.

Note that the customer folder (TAC, The Adventure Company) is placed before the layouts folder. This is so other layout related files can be stored under this folder in the future. This allows you to group all presentation details of one customer in the same folder.

If you look at the content tree structure in Sitecore, you see the following structure:



All definition items related to presentation are stored in the layout folder. This folder is a child item of the Sitecore root item. Layout definition items are stored under layout > Layouts. As a recommended practice, you should mimic the same folder structure that you define in the file system.

To create a folder structure in your project:

Detailed Steps

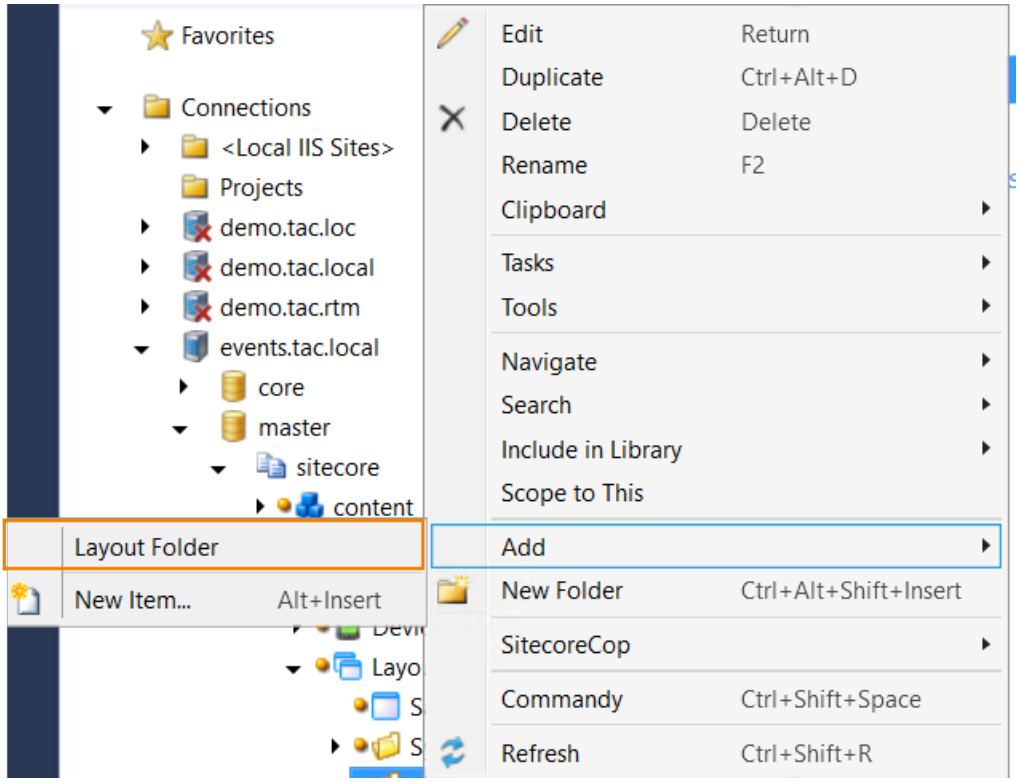
1. Using the Visual Studio Solution Explorer, in the **Views** folder, create a new folder named: *TAC*.

2. In the **TAC** project folder, create a folder named: *Events*.
3. In the **Events** project folder, create a folder named: *Layouts*.

To create a folder structure in the Sitecore tree:

Detailed Steps

1. In the Sitecore Explorer navigate to */sitecore/layout/Layouts/*, and right-click the **Layouts** folder.
2. From the context menu, choose **Add, Layout Folder**.



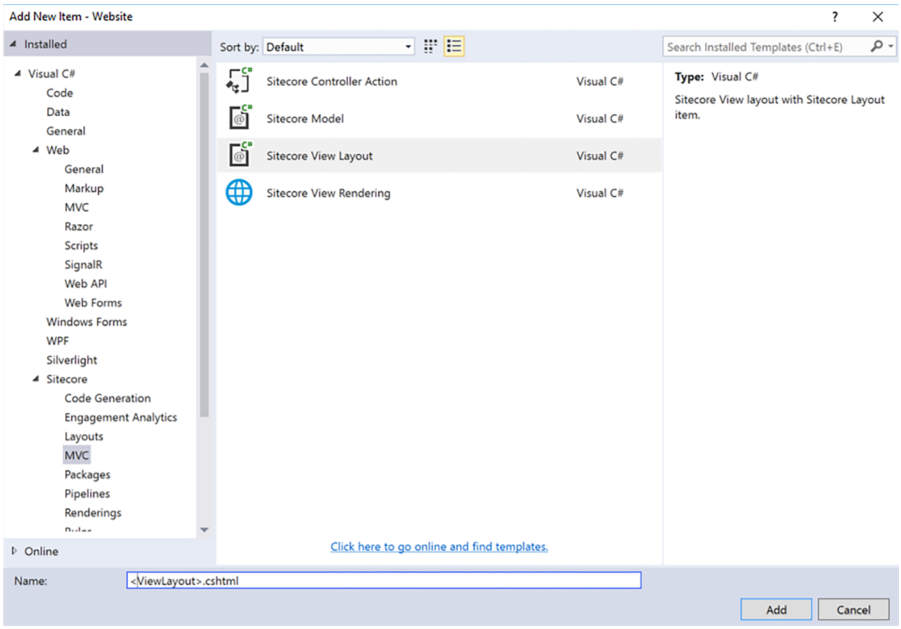
3. Name the folder: *TAC* and click **OK**.
4. Repeat to create a new Layout Folder named *Events* inside the **TAC** folder.

2. Creating an MVC Layout

Now you will create a view. This is the type of code file that is used in an MVC solution. Therefore a layout code file that is used in an MVC approach can be referred to as a *view layout*. The view layout is used to define what will be rendered to the browser. You will also create an MVC Layout definition item; its path field must contain the location of the view layout file in the file system. The code file and the definition item together are referred to as a layout or an MVC **layout**.

Detailed Steps

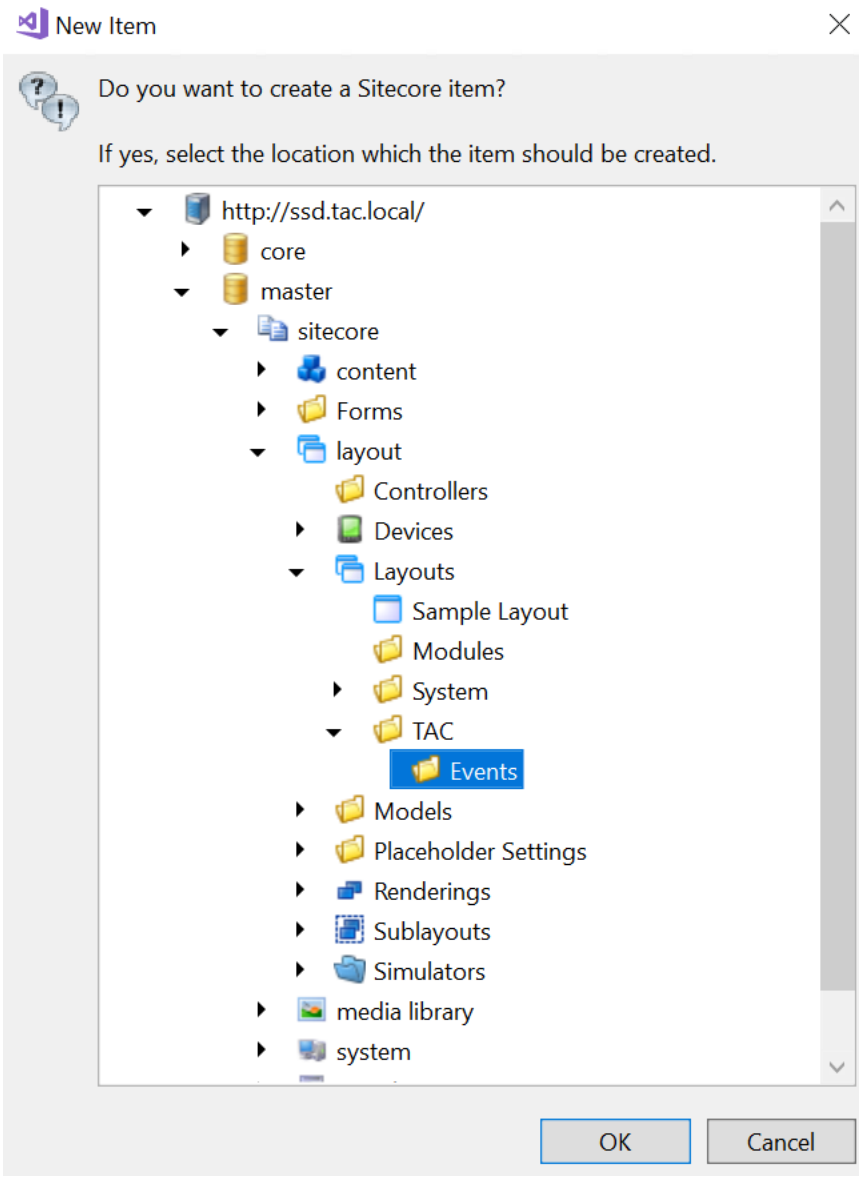
1. Right-click the **Views/TAC/Events/Layouts** folder. Choose **Add, New Item...**. In the Sitecore MVC section, select **Sitecore View Layout**.



> Sitecore Rocks has added many productivity enhancing templates into Visual Studio, for example, a view layout. This is a layout that is based on a standard MVC view. Many of these contain stub code and will often create definition items and relevant config files if your new objects need them.

2. Name the view layout: *Events.cshtml* and then click **Add**.

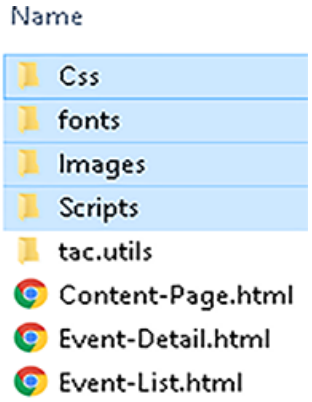
3. If Sitecore Rocks is correctly installed and connected to your project the **New Item** dialog box will appear. This dialog will automatically create the layout definition item at the selected location. Navigate to **/sitecore/layout/Layouts/TAC/Events** in the **Master db** as the target for your layout definition item. then click **OK**.



> **NOTE:** If the pop up window does not appear, you have not associated a Sitecore Rocks connection to your Visual Studio project properly. Please ensure you have completed the lab, *Setting up Sitecore Rocks*, in Module 1.

4. Notice that the view already has some predefined stub code. Copy *@Html.Sitecore().VisitorIdentification()* from the stub code and paste it above the doctype tag.

5. From the Student Resources folder, open **Event-Detail.html** in a text editor and copy all of its code.
6. Remove the default HTML code from **Events.cshtml** and paste it in the HTML from the Event-Detail.html file that you just copied.
7. Move the *@Html.Sitecore().VisitorIdentification()* method back into the head tag of the HTML of the view layout.
8. Copy the **Css, Images, Fonts** and **Scripts** folders from the Student Resources folder into the Visual Studio project (use Visual Studio's **Show All Files** and **Include in Project** features).



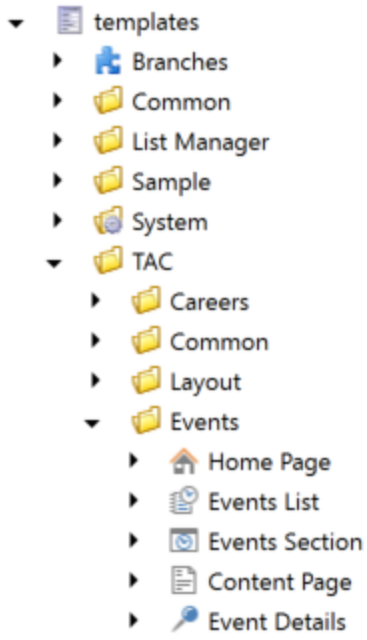
9. Deploy these files to the webroot of the Sitecore instance. You can select the folders in Visual Studio's **Solution Explorer**, right-click and select **Publish selected files**.
10. Right-click on the Views folder and select **Publish Views** (or use the shortcut Alt+;Alt+P)
11. References to the stylesheets, images and JavaScript files are relative to the current folder in the HTML-templates. In the Events.cshtml file, replace these references with a relative reference to the site. For example: *Css/vendor.min.css* becomes */Css/vendor.min.css*. You can use global search and replace, look for *"Css/* and replace for *"/Css/*. Do the same with *"Images/* and *"Scripts/*.
12. **Save** your changes and deploy the file by right-clicking it in Solution Explorer and choosing **Publish Events.cshtml**.

3. Setting presentation details on the template standard values

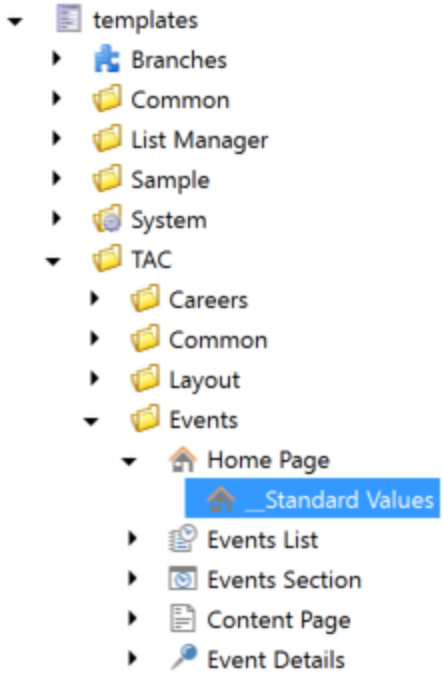
Next, you will assign the Events layout to all the items of the site. The fastest way to set the presentation for all existing items and items that will be created in the future is to set the **standard values** of a template.

Detailed Steps

1. In **Sitecore Explorer**, navigate to the **Home Page template**: */sitecore/templates/TAC/Events/Home Page*.



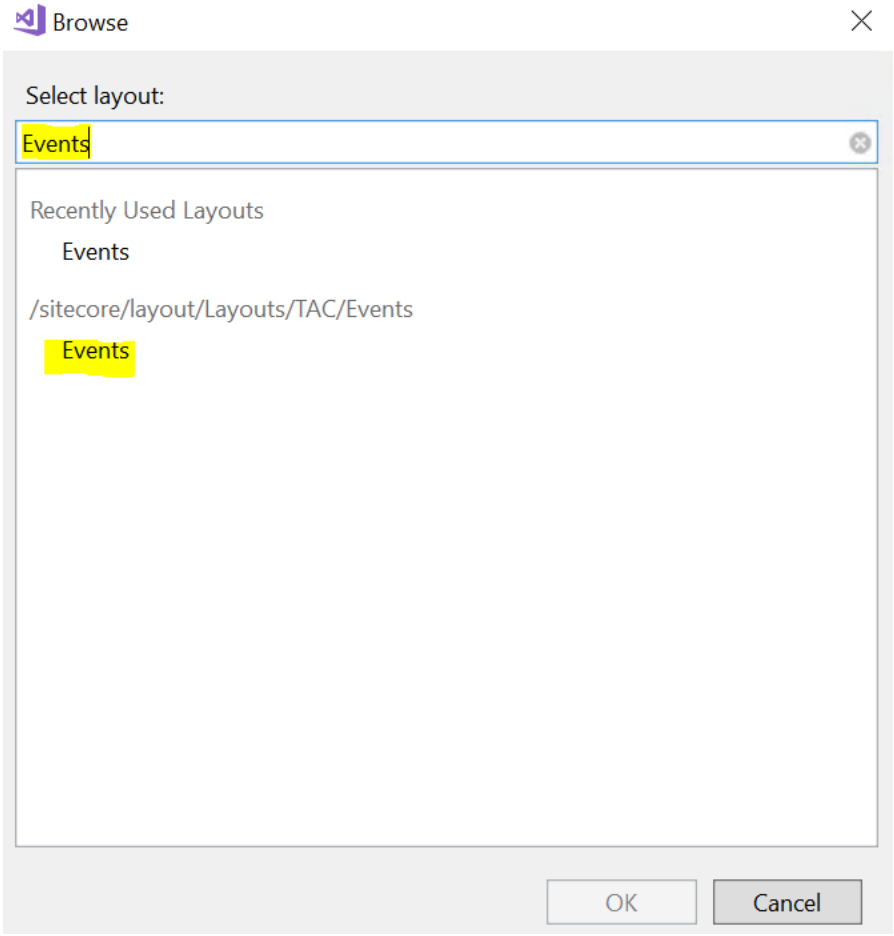
2. Expand the **Home Page** template and select its **Standard Values** item.



NOTE: Make sure you select the Standard Values item and not the template itself.

3. Right-click the **Standard Values** item and choose: **Tasks, Design Layout.**

4. On the Standard Values Layout tab, in the Layout field, click **Browse** to open the **Browse** dialog.
5. In the **Browse** dialog, select **Events**, then click **OK**.

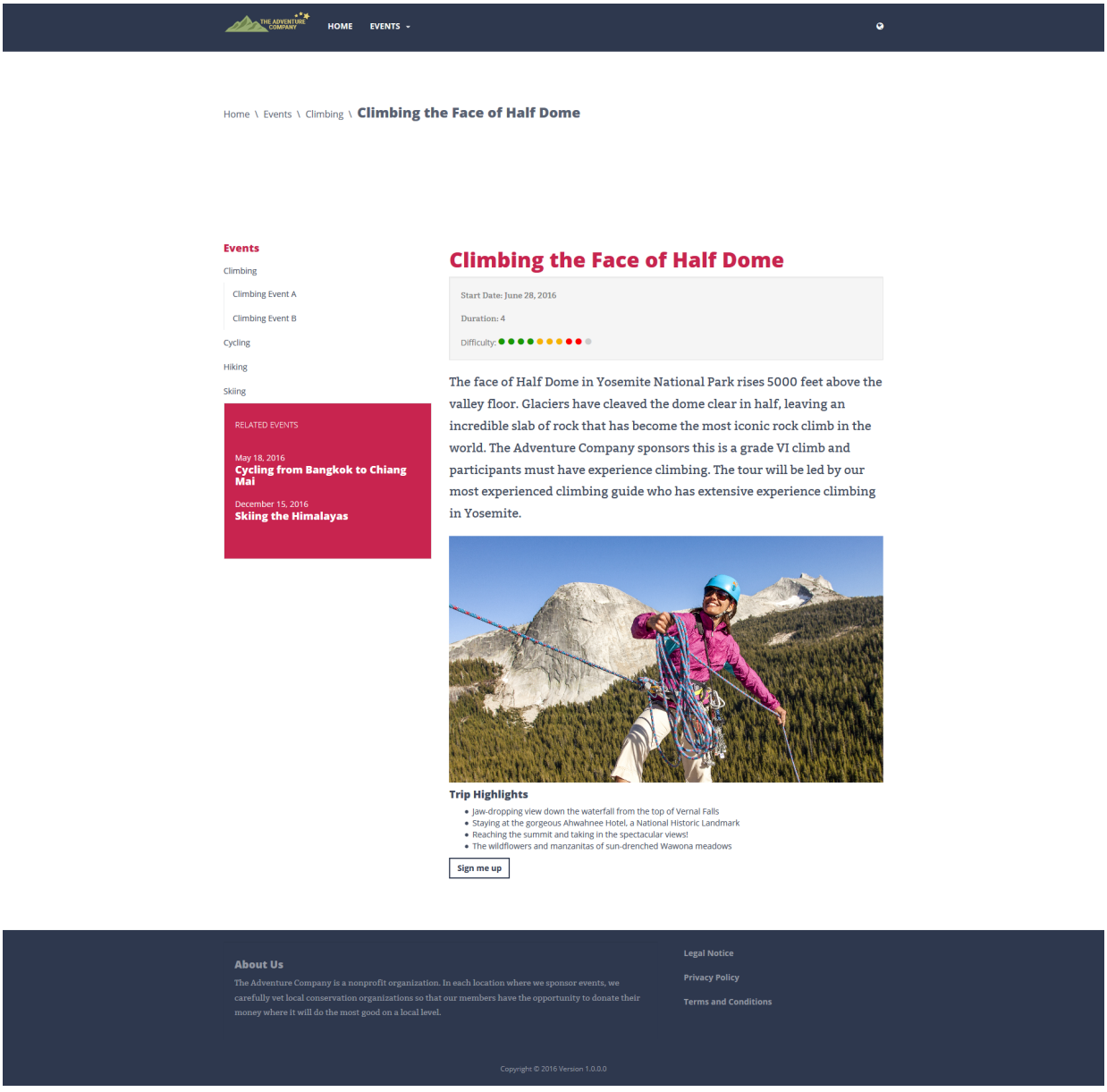


You can use the filter box to search for your layout. The filter will not take the path into consideration. A prefix on your layout might be useful when you create your layouts.

6. **Save** your changes and close the **Editor** tab.

7. Repeat the steps above to assign the *Events layout* to the standard values of the other templates representing pages in the Events website:
    - Content Page
    - Events List
    - Events Section
    - Event Details
8. To view the presentation details of an item in the Sirecore client, remove the namespace of the project from the Views/web.config. You do this because you have yet to create any classes in Visual Studio and so an attempt to render any presentation will throw an error.
   Navigate to the webroot of your site (C:\inetpub\wwwroot\tac.corporate. Open the **Views/web.config**, and remove the line declaring: **<add namespace="events.tac.local" />**

9. Select the **Event Details** item, */sitecore/content/Home/Events/Climbing/Climb Mount Everest* and preview the item in a browser. You can preview an item by right-clicking the item in the Sitecore Explorer and select Tools, Browse, **Preview**.



STOP HERE

Lab: Implementing Inline Editing

In this lab, you will learn how to render the content of fields using the Sitecore field helper. When you use the field helper, the content is not only displayed, you can also edit it in the Experience Editor. You will make most of the content for an event both visible and editable.

1. Rendering Content from fields

Detailed Steps

1. In the *Views/TAC/Events/Layouts* folder, open **Events.cshtml**.

2. To render the existing content for the different fields, locate the HTML for each of the specified fields in the view and replace the xxx with the **Sitecore field helper**. Be aware that some fields need a parameter to render correctly.

Field	HTML Location	Parameters
MetaDescription	<meta name="description" content="*xxx*" />	DisableWebEdit=true
MetaTitle	<title>*xxx*</title>	DisableWebEdit=true
ContentHeading	*xxx*	
StartDate	Start Date: *xxx*	format="dd MMM yyyy"
Duration	Duration: *xxx*	
ContentBody	*xxx*	
EventImage	*xxx*	mw=1000, mh=568, @class="img-responsive"
Highlights	Trip Highlights xxx	

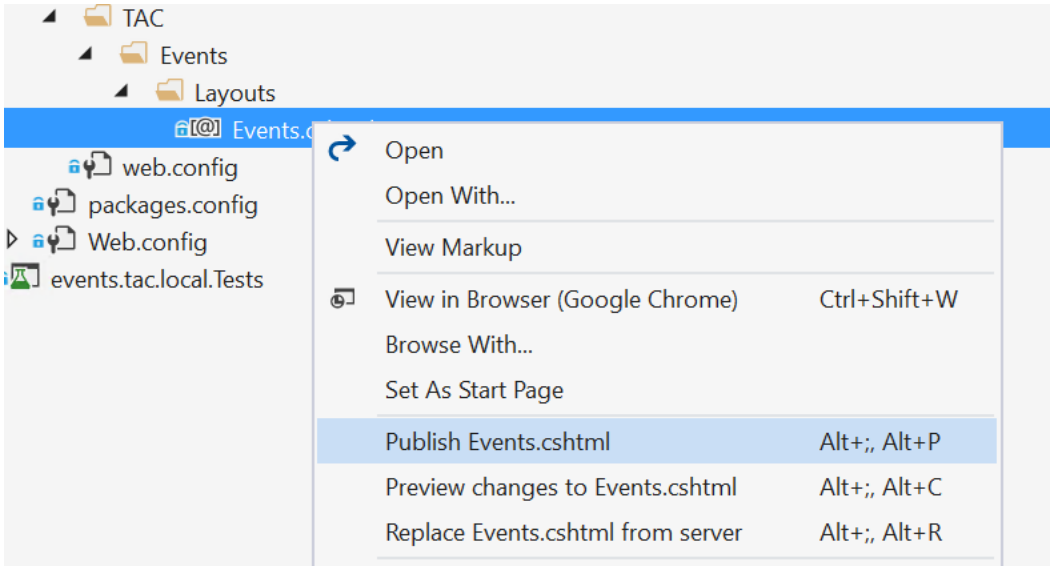
NOTE: Instead of the field name, you can also use the ID of the field item, which is a child element of the template. These field ID's can be stored as constants in a static class.

The following is the result for the Event Content section in the HTML.

```
<!-- [Event Content] -->
<div class="lead">
    @Html.Sitecore().Field("ContentBody")
</div>
<div class="article image">
    @Html.Sitecore().Field("Event Image", new { mw=1000, mh=568, @class="img-responsive" })
</div>
<div class="highlights">
    <h4>Trip Highlights</h4>
    @Html.Sitecore().Field("Highlights")
</div>
<!-- [/Event Content] -->
```

NOTE: You pass parameters to the field helper by using an anonymous object.

3. **Save** your changes.
4. Deploy your code changes using the **Publish** file command on **Solution Explorer** content.

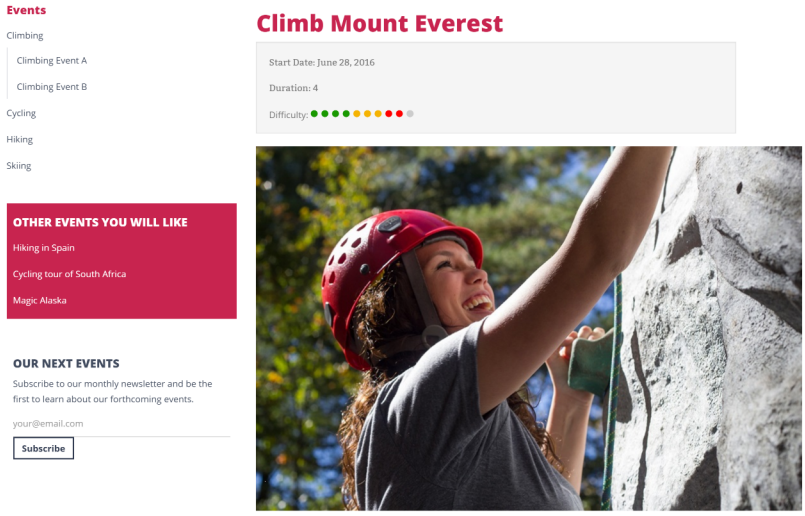


2. Preview and edit a page

Detailed Steps

1. Using the Sitecore Explorer in Sitecore Rocks, in the **Events site** folder, navigate to an Event Details item, for example, *Home/Events/Climbing/Climb Mount Everest.*
2. Open the item and ensure the **ContentHeading**, **EventImage** and **MetaTitle**fields all have values in them. Save any changes you make.

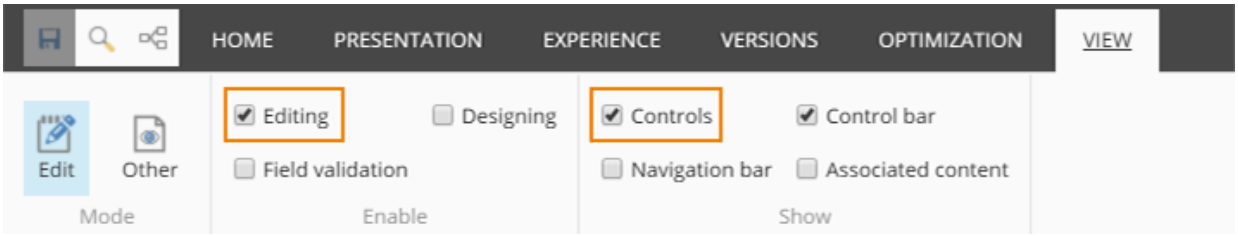
3. Right-click the **Event Details** item and, from the Context menu, choose Tools, Browse **Preview**.



4. Switch to the **Experience Editor** by choosing **Edit** on the ribbon. If the ribbon is collapsed use the **Show/Hide** arrow on the right side of the ribbon to open it. img src="../../../Resources/Images/Technical Training/SXSD_EE_RibbonEditShowHide.png" />

   To switch from **Preview** mode to the **Experience Editor,** click **Edit**. The ribbon can be expanded or collapsed from the **Show/Hide** arrow on the right.

5. On the **View** tab, ensure that **Editing** and **Controls** are enabled.



6. Click the **ContentHeading** to view its floating toolbar.

7. Change the **ContentHeading** field to: "*Exciting Mount Everest Climb!*" then **save** your changes.

8. Close the **Experience Editor** tab in the browser.

STOP HERE

Lab: Creating a Component

In this lab, you will create a controller rendering consisting of a model, a controller, and a view. You will statically add the component to the layout.

1. Create a model

The model retrieves content, in this case, for an event. In this section, you will build a model, a controller to handle requests for the model, and a view to display the model.

Detailed Steps

1. In your Visual Studio **events.tac.local** project, right-click the **Models** folder and choose **Add, Class...** from the context menu. Name the new class: *EventIntro.cs*. Click **Add** to create the class.
2. Add a public property of the type *HtmlString* named *ContentHeading* with empty getter and setters. Use the following code as a guide:

```
public HtmlString ContentHeading { get; set; }
```

public HtmlString ContentHeading { get; set; }

> You are using HtmlString as a property type because this property will contain HTML and this will ensure it gets appropriately rendered. If you used a string type it would get escaped when rendered on the view.

3. Repeat the previous step to create properties for the remaining fields.

PropertyType	Property Name
HtmlString	ContentIntro
HtmlString	ContentBody
HtmlString	EventImage
HtmlString	Highlights
HtmlString	StartDate
HtmlString	Duration
HtmlString	DifficultyLevel

Your completed code looks like this:

```
public class EventIntro
{
    public HtmlString ContentHeading { get; set; }
    public HtmlString ContentIntro { get; set; }
    public HtmlString ContentBody { get; set; }
    public HtmlString EventImage { get; set; }
    public HtmlString Highlights { get; set; }
    public HtmlString StartDate { get; set; }
    public HtmlString Duration { get; set; }
    public HtmlString DifficultyLevel { get; set; }
}
```

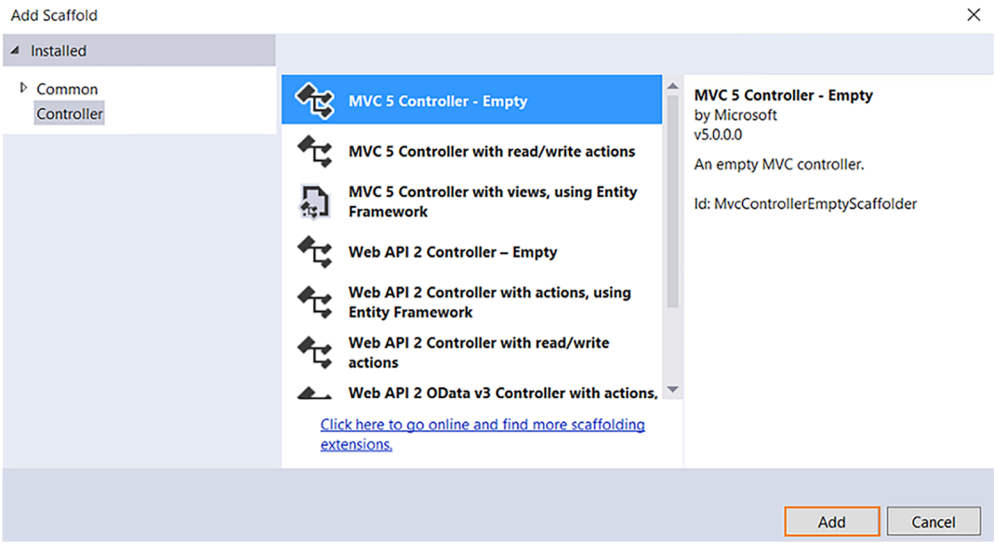
4. **Save** your work and close the tab.

2. Adding a controller

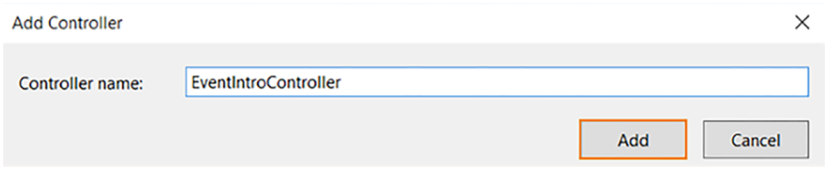
Detailed Steps

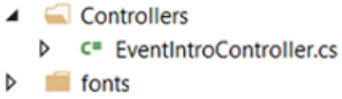
1. Right-click the **Controllers** folder and select **Add, Controller...**

2. In the **Add Scaffold** dialog select: **MVC 5 Controller - Empty** and then click **Add**.

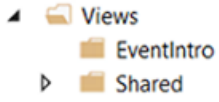


3. Name the controller: *EventIntroController* and then click **Add**.





NOTE: When you create the controller, matching folders are created in the Views folder.



4. Write a **CreateModel()** method to initialize the model. You will use the FieldRenderer to make your fields editable in the Experience Editor. To be able to use the FieldRenderer in your view, add a using directive to *Sitecore.Web.UI.WebControls*. Be sure to also add references to the Model's namespace and *Sitecore.Mvc.Presentation* so that you can access the **RenderingContext** object:

```
public static EventIntro CreateModel()
{
    var item = RenderingContext.Current.ContextItem;
    return new EventIntro()
    {
        ContentHeading = new HtmlString(FieldRenderer.Render(item, "ContentHeading")),
        ContentBody = new HtmlString(FieldRenderer.Render(item, "ContentBody")),
        EventImage = new HtmlString(FieldRenderer.Render(item, "EventImage", "mw=400")),
        Highlights = new HtmlString(FieldRenderer.Render(item, "Highlights")),
        ContentIntro = new HtmlString(FieldRenderer.Render(item, "ContentIntro")),
        StartDate = new HtmlString(FieldRenderer.Render(item, "StartDate")),
        Duration = new HtmlString(FieldRenderer.Render(item, "Duration")),
        DifficultyLevel = new HtmlString(FieldRenderer.Render(item, "DifficultyLevel"))
    };
}
```

```
public static EventIntro CreateModel()
{
       var item = RenderingContext.Current.ContextItem;
       return new EventIntro()
       {
                ContentHeading = new HtmlString(FieldRenderer.Render(item,
"ContentHeading")),
                ContentBody = new HtmlString(FieldRenderer.Render(item, "ContentBody"
                EventImage = new HtmlString(FieldRenderer.Render(item, "EventImage",
"mw=400")),
                Highlights = new HtmlString(FieldRenderer.Render(item, "Highlights"))
                ContentIntro = new HtmlString(FieldRenderer.Render(item, "ContentIntr
                StartDate = new HtmlString(FieldRenderer.Render(item, "StartDate")),
                Duration = new HtmlString(FieldRenderer.Render(item, "Duration")),
                DifficultyLevel = new HtmlString(FieldRenderer.Render(item,
"DifficultyLevel"))
       };
}
```

5. Pass the model into the view. Do this by passing the model as parameter to the View method in the return statement of the **Index** method :

```
// GET: EventIntro
public ActionResult Index()
{
    return View(CreateModel());
}
```

return View(CreateModel());

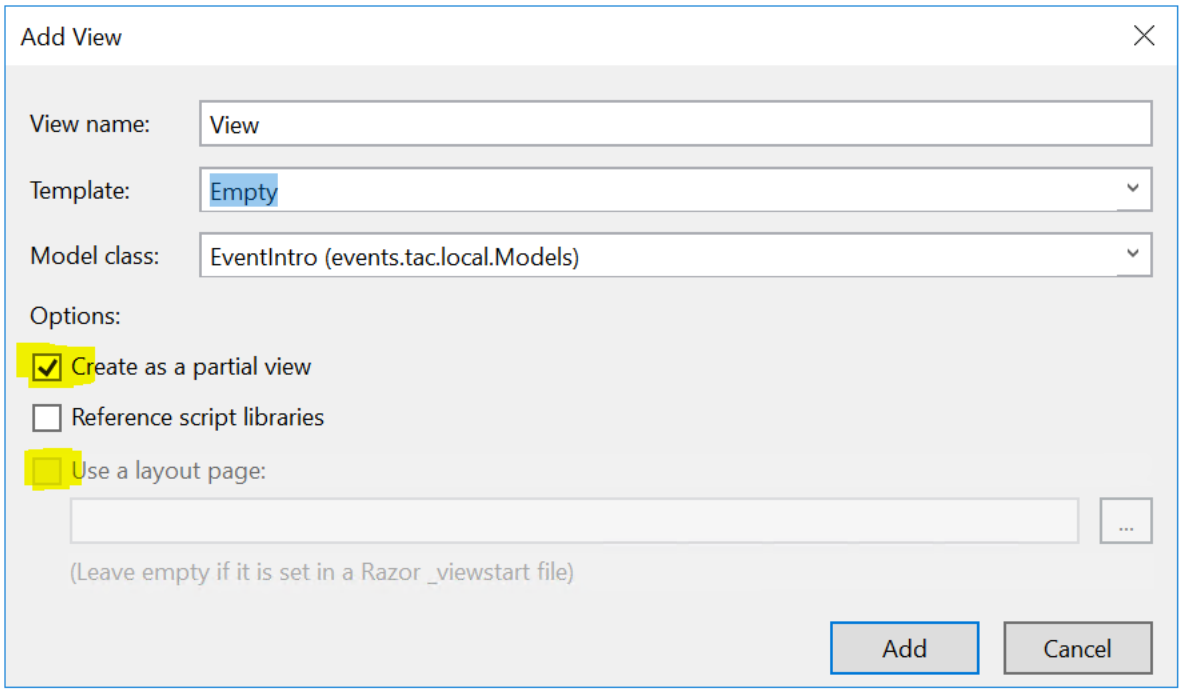
6. **Save** your work and close the Editing tab.

3. Creating a view

Detailed Steps

1. Right-click the **Views/EventIntro** folder and choose: **Add, View...**.

2. Complete the dialog:
    a. In the **View name** field, type: *Index.*
    b. In the **Template** drop-down list, select: **Empty.**
    c. In the **Model class** field, type: "*E*" and let auto-complete find the *EventIntro class*.
    d. Ensure that the **Use a layout** page option is unchecked. If the option is disabled (greyed out), you may have to disable **Create as a partial view** first.
    e. Select the **Create as a partial view** check box.
    f. Click **Add** to create the view.



3. Ensure your view already has a reference to the **EventIntro** model:

```
@model events.tac.local.Models.EventIntro
```

@model events.tac.local.Models.EventIntro

4. **Open** */Views/TAC/Events/Layouts/Events.cshtml* and cut the contents of the section commented: <!--[Event Info] --> and then paste it into the view.

5. In the view (*Index.cshtml*), replace the fields that you have previously rendered with the field helper with the equivalent form the model:

Field Helper	Model
@Html.Sitecore().Field("ContentHeading")	@Model.ContentHeading
@Html.Sitecore().Field("Start Date", new { format = "dd MMMM yyyy" })	@Model.StartDate
@Html.Sitecore().Field("Duration")	@Model.Duration
@Html.Sitecore().Field("ContentBody")	@Model.ContentBody
@Html.Sitecore().Field("EventImage", new { mw = 1000, mh = 568, @class = "img-responsive" })	@Model.EventImage
@Html.Sitecore().Field("Highlights")	@Model.Highlights

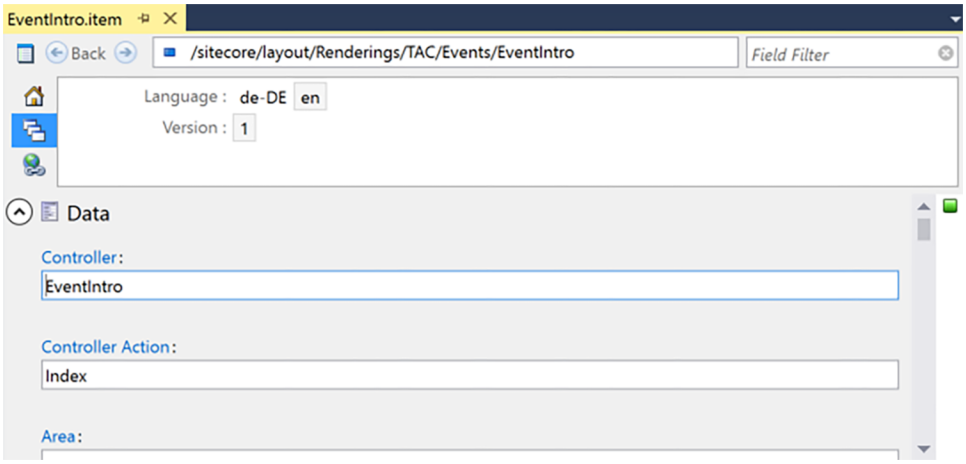
6. **Save** your changes.

4. Adding a controller rendering definition item

Detailed Steps

1. In the **Sitecore Explorer**, add a new item */sitecore/layout/Renderings/TAC/* that is based on the template *Rendering Folder* (it is one of the Insert Options).
2. Add another Rendering Folder named *Events* inside the TAC folder.
3. Right-click the *Renderings/TAC/Events* folder and choose Add, **Controller rendering**. Name the rendering: *EventIntro* and click **OK**.

4. In the **Controller** field, type: *EventIntro*. In the **Controller Action** field, type: *Index*.



> **NOTE:** Although the file name is *EventIntroController.cs*, the text "*controller*" is not needed in the item's Controller field because it is understood.

5. **Save** your changes and close the Editor tab.

5. Binding a component statically

Detailed Steps

1. Switch back to your layout code file *Views/TAC/Events/Layouts/Events.cshtml* file and scroll to the .
2. **Type** the following in the Event Info section to **statically** include the **Event Intro** rendering:

```
<div class="col-lg-8">
    <!-- [Event Info] -->
    @Html.Sitecore().Rendering("/sitecore/layout/Renderings/TAC/Events/EventIntro")

    <!-- [/Event Info] -->
```

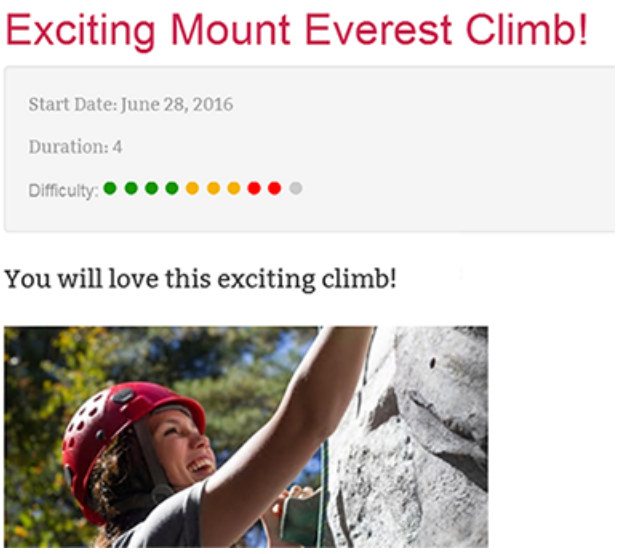
@Html.Sitecore().Rendering("/sitecore/layout/Renderings/TAC/Events/EventIntro")

6. Deploy and preview your changes

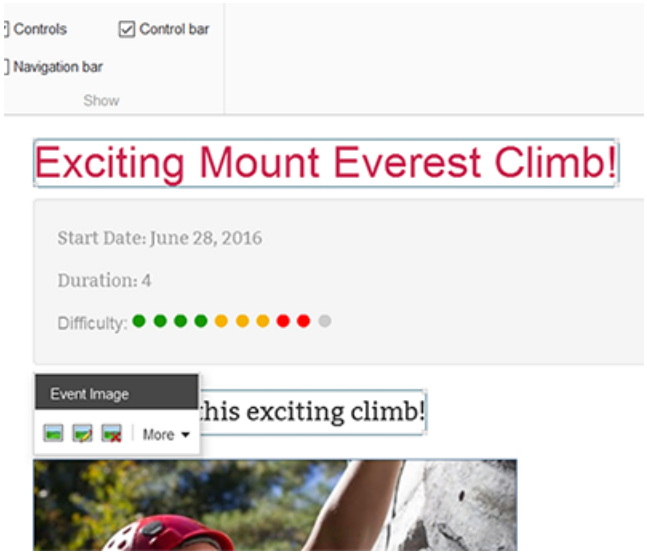
Detailed Steps

1. On the Visual Studio toolbar, **deploy** your code by clicking **Web One Click Publish**.

2. Preview an **Event Details** item, for example, */sitecore/content/Home/Events/Climbing/Climb Mount Everest*. Right-click and choose: Tools, Browse, **Preview.**



3. Switch to the **Experience Editor** and ensure **Editing** is on and check that your content is still editable.



> **NOTE:** If you did not disable the **Use a layout page** option during scaffolding, some shared views are created. These views conflict with how controller markup is rendered. To avoid the conflict, comment out the code in the file /Views/_ViewStart.cshtml.

STOP HERE

Lab: Componentizing the Website

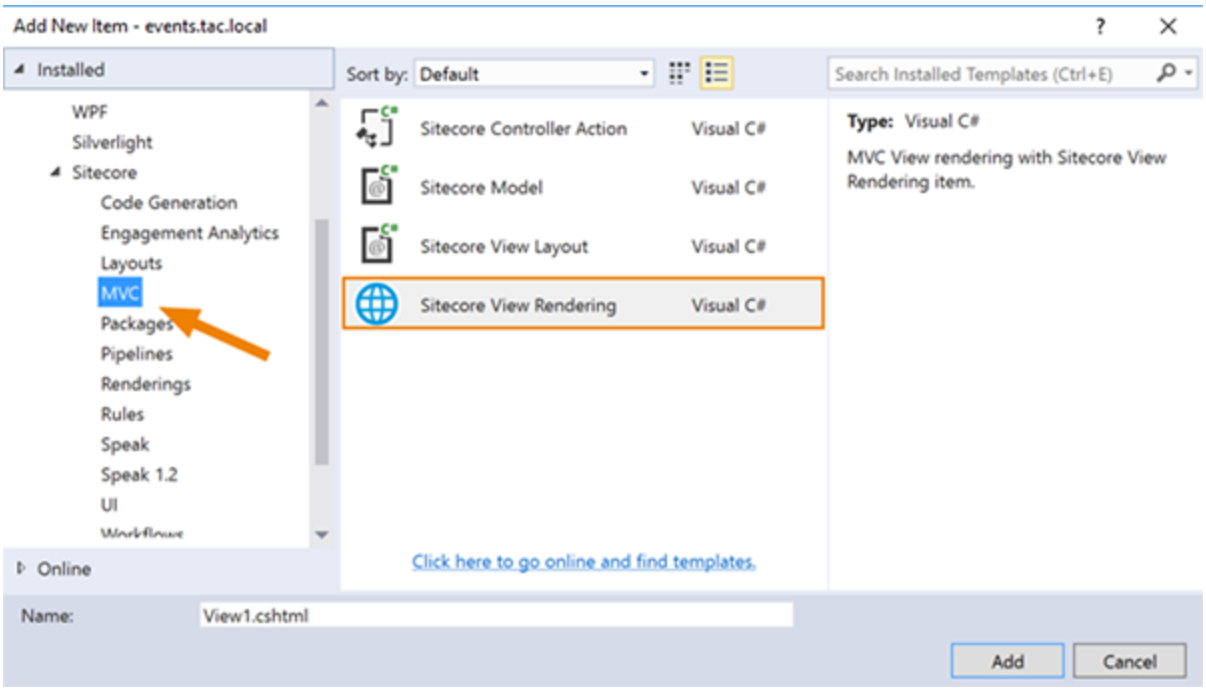
In this lab, you will move static markup from the layout onto components. You will then add placeholders and configure the presentation details to add the components back on the page using dynamic binding.

1. Creating components and adding placeholders

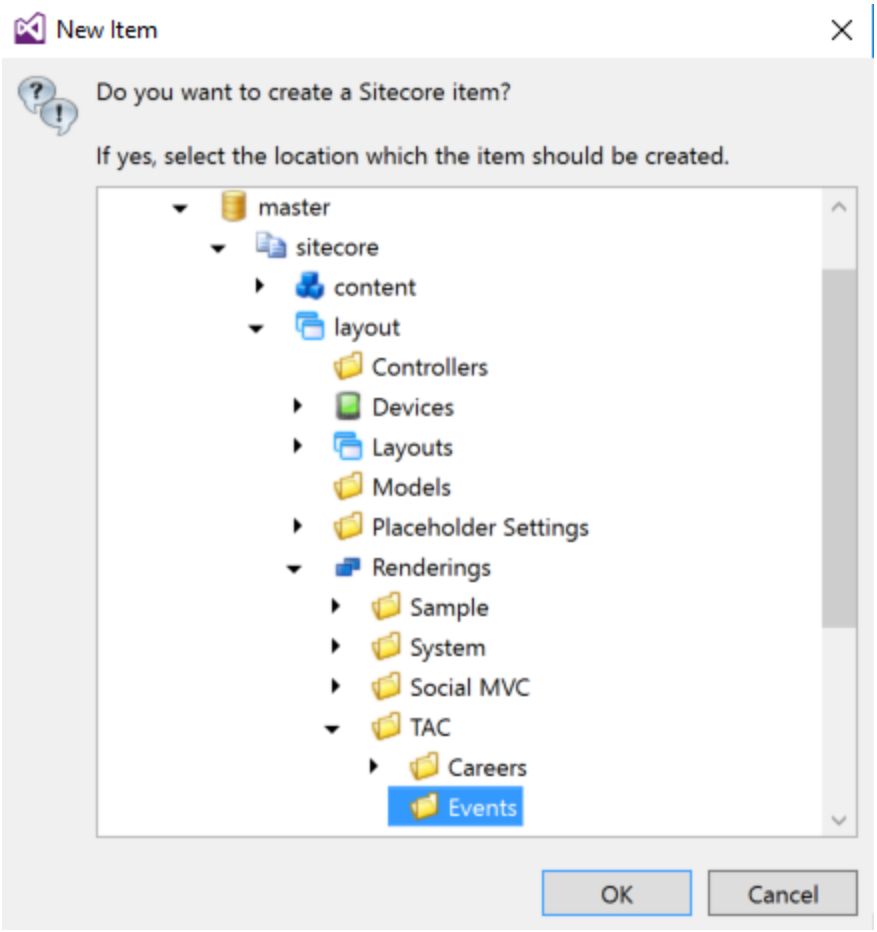
Detailed Steps

1. In **Solution Explorer**, in the *Views/TAC/Events* folder, create a new Sitecore View Rendering, and name it: *SideContent.cshtml.*

> **NOTE:** View renderings do not need a model or controller, and can be created directly in Visual Studio by using *Add, New Item, SItecore, MVC, Sitecore View Rendering*.



2. When you are prompted select */sitecore/layout/Renderings/TAC/Events* as the location for the definition item.



> **NOTE:** If the New Item dialog box does not appear automatically, your Sitecore Rocks connection is not connected to your Visual Studio Project (or you may have local installation issues). In this case, you have to create the definition items manually.

3. Remove the empty div.
4. In **Solution Explorer**, open the */Views/TAC/Events/Layouts/Events.cshtml* layout.
5. Scroll down to the section labeled: <!-- [Two Columns] -->
6. Under the **Two Columns** section, find the second which looks like: . Cut everything inside the **<aside>** node and paste into **SideContent.cshtml**.
7. **Save** the view rendering and deploy the file.
8. In **Solution Explorer**, in the ***Views/TAC/Events*** folder, create a new Sitecore View Rendering, and name it: *PageContent.cshtml.*
9. Remove the divs.
10. Open the **Events.cshtml** layout file and scroll down to the section labelled: .

11. Delete the statically rendered **EventIntro** from the markup below.
12. Select the remaining markup inside the div with the class col-lg-8.
13. Cut and paste the code into **PageContent.cshtml**.
14. **Save** the view rendering and deploy the file.
15. In **Events.cshtml**, the **Two Columns** section should now look like this:

```
<!-- [Two Columns] -->
<div class="row">
    <div class="col-lg-4">
        <aside>

        </aside>
    </div>
    <div class="col-lg-8">

    </div>
</div>
<!-- [/Two Columns] -->
```

> **NOTE:** The HTML templates are built using the Bootstrap grid system. This template uses a two column grid that consists of a left column that is half the size of the right column. More information on the bootstrap grid system can be found at: *https://getbootstrap.com/examples/grid/*

16. Replace </aside> tags with @Html.Sitecore().Placeholder(). For each column you will specify a placeholder. In the smaller column (col-lg-4), add a placeholder with the name *pageSide*. In the wider column (col-lg-8), add a placeholder with the name *pageContent*.

```
<!-- [Two Columns] -->
<div class="row">
    <div class="col-lg-4">
        @Html.Sitecore().Placeholder("pageSide")
    </div>
    <div class="col-lg-8">
        @Html.Sitecore().Placeholder("pageContent")
    </div>
</div>
<!-- [/Two Columns] -->
```


<!-- [Two Columns] -->

@Html.Sitecore().Placeholder("pageSide")

@Html.Sitecore().Placeholder("pageContent")

<!-- [/Two Columns] -->

17. Delete the markup for the breadcrumb navigation and replace this with a placeholder called *pageHeader*.

> **NOTE:** Later you will create a breadcrumb component and copy the code back in from the HTML templates.

```
<!-- [Page Content] -->
<main>
    <header class="page-header">
        <div class="container">
            @Html.Sitecore().Placeholder("pageHeader")
        </div>
    </header>
```

18. **Save** your changes.

2. Create the two-column component

Detailed Steps

1. In **Solution Explorer**, in the **Views/TAC/Events** folder, create a new Sitecore View Rendering.
2. Name the rendering: *Two-Column.cshtml.*
3. When you are prompted select */sitecore/layout/Renderings/TAC/Events* as the location for the definition item.
4. Remove the empty div from the file.
5. In **Solution Explorer**, open the **Events.cshtml** layout.
6. Cut the markup inside the <main> tags and replace it with a placeholder called *main*.

```
<!-- [Page Content] -->
<main>
    @Html.Sitecore().Placeholder("main")
</main>
<!-- [/Page Content] -->
```

7. Paste the markup in the Two-Column.cshtml.

```
<header class="page-header">
    <div class="container">
        @Html.Sitecore().Placeholder("pageHeader")
    </div>
</header>
<section class="page-header">
    <div class="container">
        <!-- [Two Columns] -->
        <div class="row">
            <div class="col-lg-4">
                @Html.Sitecore().Placeholder("pageSide")
            </div>
            <div class="col-lg-8">
                @Html.Sitecore().Placeholder("pageContent")
            </div>
        </div>
        <!-- [/Two Columns] -->
    </div>
</section>
```

8. **Save** all the changes and deploy both files (Two-Column.cshtml and Events.cshtml).

3. Create the one-column component

Detailed Steps

1. In **Solution Explorer**, in the *Views/TAC/Events* folder, create a new Sitecore View Rendering.
2. Name the rendering: *One-Column.cshtml.*
3. When prompted select */sitecore/layout/Renderings/TAC/Events* as the location for the definition item.
4. Remove the empty div from the file.
5. Open the **Content-Page.html** file from the Student Resources folder.
6. Copy the markup inside the <main> tags and paste it in the One-Column.cshtml.
7. Replace the section labeled with the **Breadcrumb Navigation** (including the comments) with a placeholder named *pageHeader*.
8. Replace the content inside the div marked with the class **col-md-12** with a placeholder named *pageContent*.
9. Remove the second div decorated with the container class inside the header tag. The code should look like this:

```
<header class="page-header">
    <div class="container">
        @Html.Sitecore().Placeholder("pageHeader")
    </div>
</header>
<section class="section section-full">
    <div class="container">
        <!-- [One Column] -->
        <div class="row">
            <div class="col-md-12">
                @Html.Sitecore().Placeholder("pageContent")
            </div>
        </div>
        <!-- [/One Column] -->
    </div>
</section>
```

10. **Save** the changes and deploy the One-Column.cshtml file.

4. Create a component with the markup for the Home page

Detailed Steps

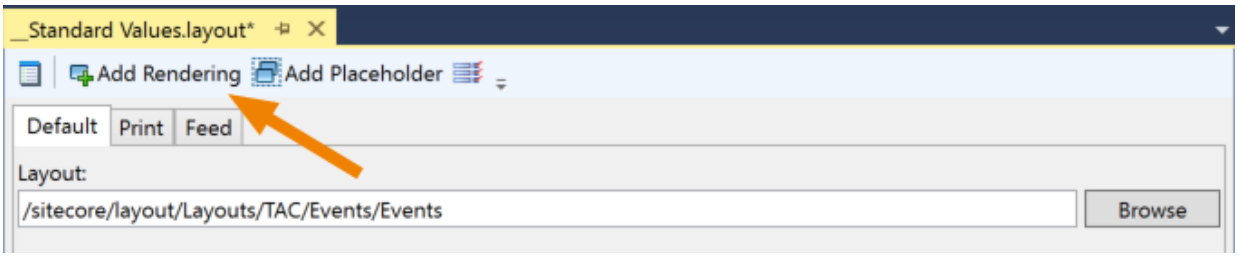
1. In **Solution Explorer**, in the *Views/TAC/Events* folder, create a new Sitecore View Rendering, and name it: *HomeContent.cshtml.*
2. When prompted select */sitecore/layout/Renderings/TAC/Events* as the location for the definition item.

3. Remove the empty div.
4. Open the **Index.html** file from the Student Resources folder.
5. Scroll down to the section labelled: <!-- [Page Content] -->
6. In the Page Content section, copy everything **inside** (but excluding) the <main> tags and paste into **HomeContent.cshtml**.
7. **Save** and **deploy** the file.

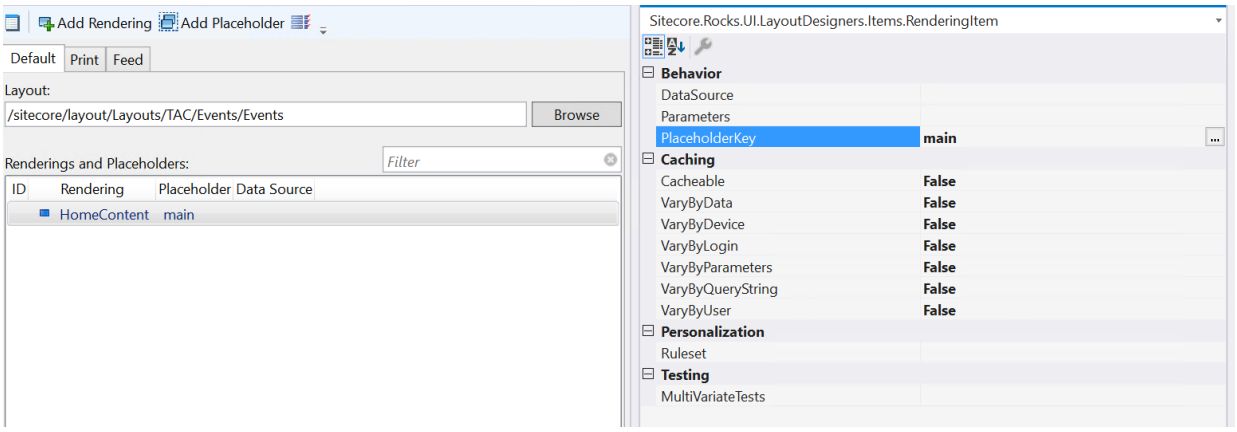
5. Setting presentation details for the Home page

Detailed Steps

1. In the **Sitecore Explorer**, select the **Home Page Template Standard Values** item. Right-click and choose: **Tasks > Design Layout**. If you are prompted, choose the *Copy and edit the layout…* command.
2. Click the **Add Rendering** command. In the **Select Renderings** dialog:
    a. Select **All** from the **Filter** list.
    b. Type *home* into the **Filter** text box.
    c. Select the **HomeContent** view rendering item from the list.
    d. Click **OK.**



3. Double-click the **HomeContent** item in the renderings list (or select it and press F4) to open the **Properties** window for the component. In the component properties, click the ellipsis in the **Placeholder Key** property and choose *main* from the list of placeholders in the Browse dialog (or type *main* into the **Placeholder Key** property). Click **OK**.



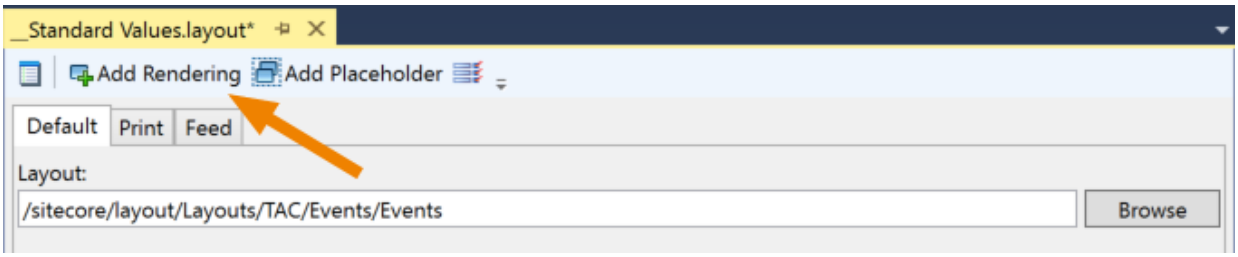
4. **Save** your changes and close the **Editor** tab.
5. Preview the Home page You can use the Sitecore Rocks command Tools, Browse, Preview.

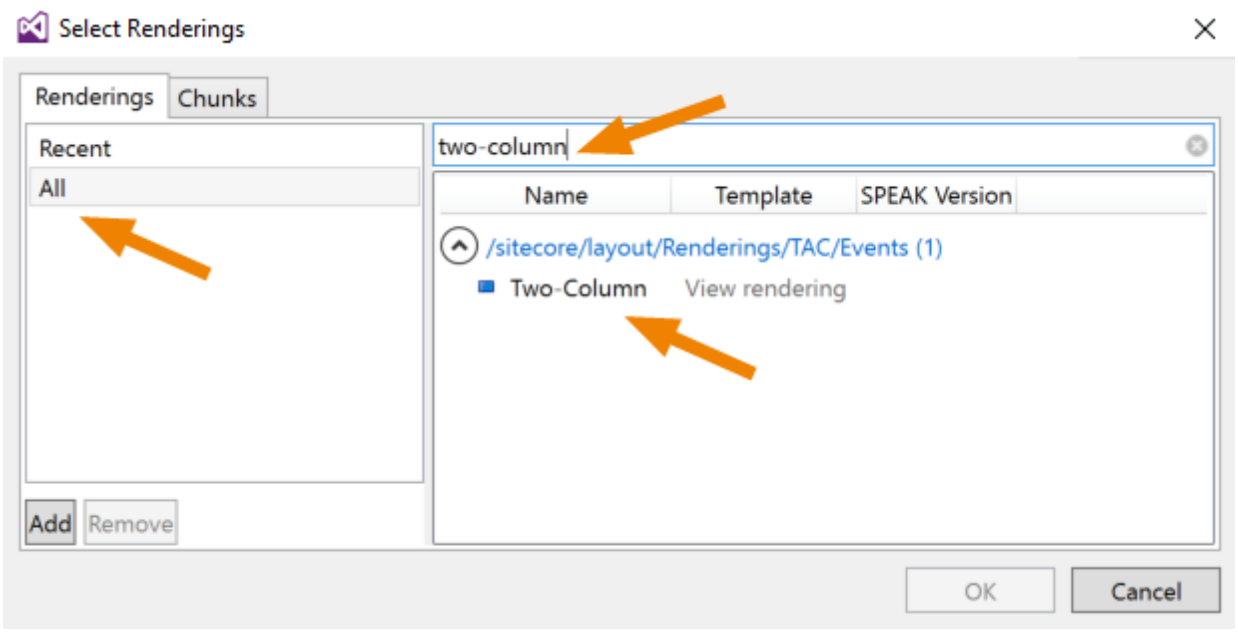
6. Setting presentation details for Event Details pages

Detailed Steps

1. In the **Sitecore Explorer**, select the **Event Details Template Standard Values** item, right-click and choose: **Tasks, Design Layout**. If you are prompted choose *Copy and edit the layout…*.

2. Click the **Add Rendering** command. In the **Select Renderings** dialog:
    a. Select **All** from the **Filter** list.
    b. Type *two-column* into the **Filter** text box.
    c. Select the **Two-Column** view rendering item from the list.
    d. Click **OK.**

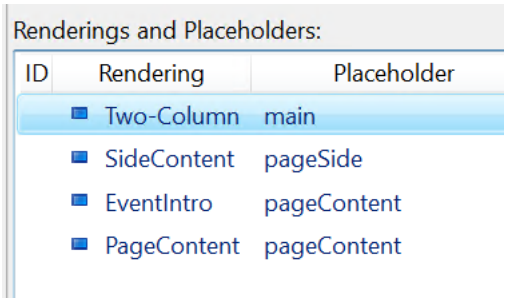




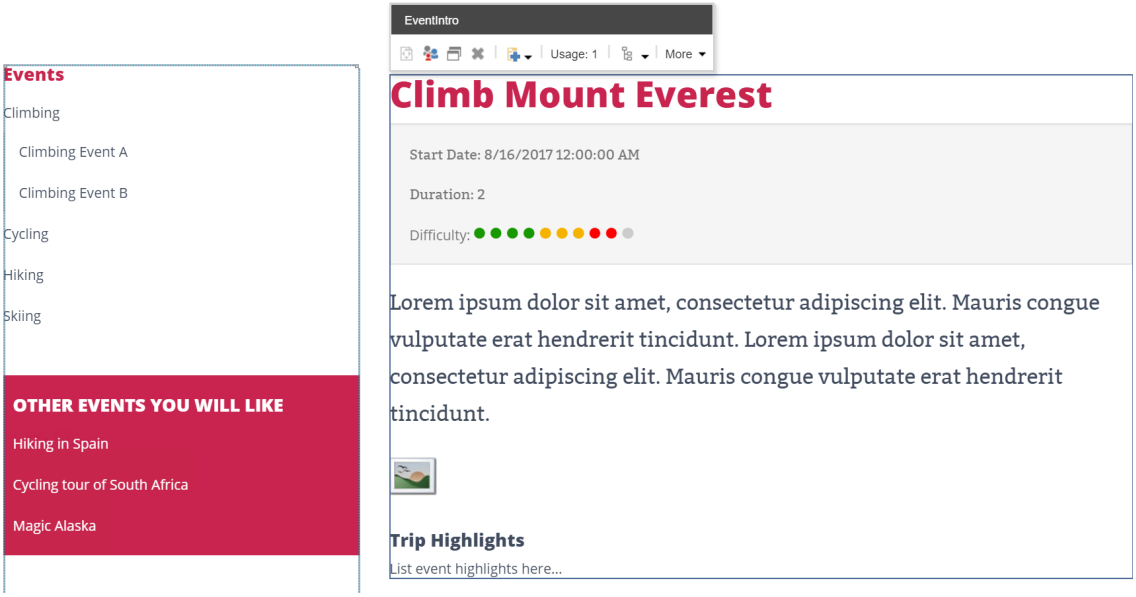
3. Assign the Two-Column rendering to the **main** placeholder.

4. Add the following components to the indicated placeholders:

Component name	Placeholder
SideContent	pageSide
EventIntro	pageContent
PageContent	pageContent



5. **Save** your changes and leave the editor tab open.
6. **Preview** an Event Details item, for example, */sitecore/content/Events/Home/Events/Climbing/Climb Mount Everest* . Notice the page looks and acts the same as when you statically rendered it
7. Use the **Edit** button to switch to the **Experience Editor** with **Designing** and **Controls** enabled. Notice now that it is the different *components* that are automatically highlighted.



Notice how the components can be selected but they can't be removed, or modified. The buttons in the toolbar are disabled. In the next lab you will configure the Experience Editor to enable this functionality.

Lab: Using Dynamic Placeholders

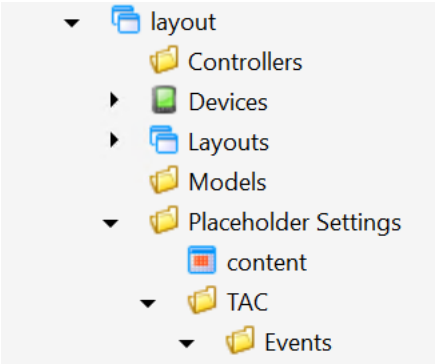
In this lab, you will discover how to use placeholder settings to control authors' permissions in the Experience Editor. You will also add dynamic placeholders to create a more flexible componentization of the Home page.

1. Adding placeholder definition items

You will now add placeholder definition items to make sure that the placeholders are selectable in the Experience Editor. In addition, you will set the allowed controls for these placeholders. This restricts the components that can be bound to the placeholders through the Experience Editor. You can always add components to any placeholder through Sitecore Rocks or presentation details for users with additional rights.

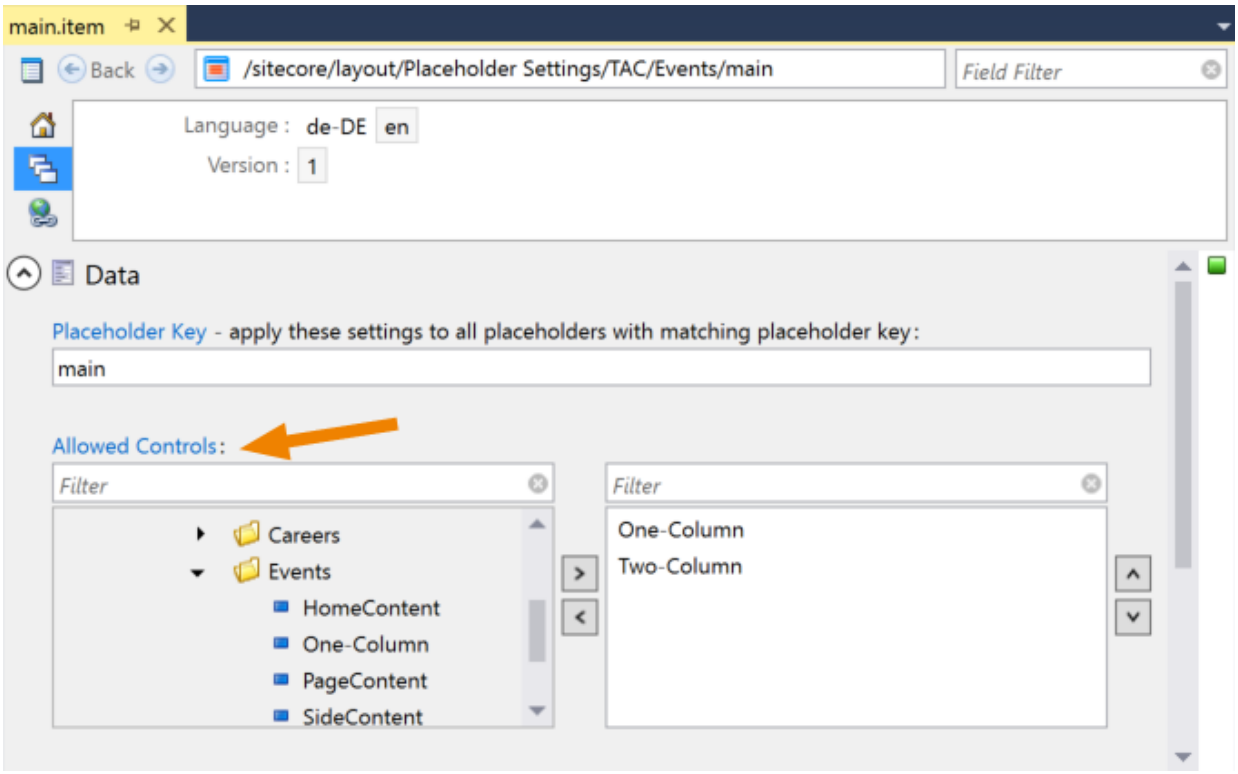
Detailed Steps

1. In the **Sitecore Explorer**, using the Placeholder Settings Folder template, create a **TAC** and **Events** folder inside the /sitecore/layout/Placeholder Settings item.



2. Right-click on the **Events** folder and choose **Add > Placeholder** to name an item named *main*.

3. Select the **One-Column** and **Two-Column** components in the **Allowed Controls** field.



4. **Save** and close the item.
5. Repeat the steps above to create placeholder settings and set the allowed controls for the following placeholders:

Placeholder	Allowed Controls
pageContent	• /sitecore/layout/Renderings/TAC/Events/PageContent
pageSide	• /sitecore/layout/Renderings/TAC/Events/SideContent
pageHeader	• N/A

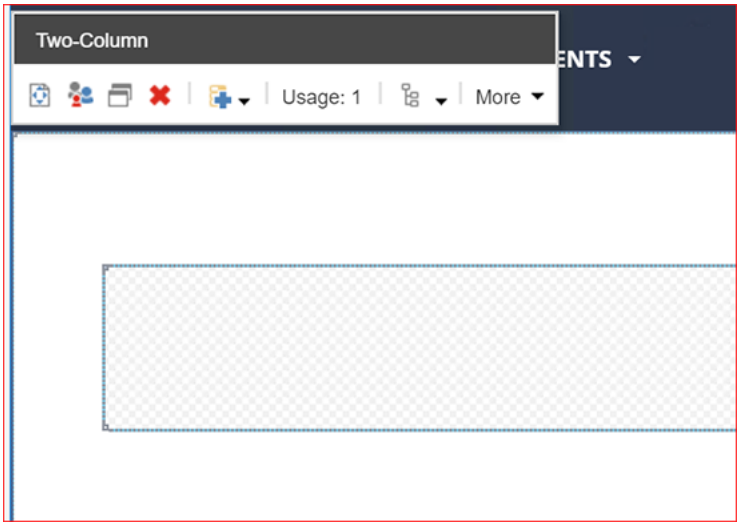
2. Using the Experience Editor to change the design of the page

Having configured the placeholder settings, an author will be able to use the Experience Editor to add and remove components. You will now try to change the presentation of an Event Details page to display in one column instead of two.

Detailed Steps

1. Preview one of the Event Detail items, for example, **Climbing Mount Everest**.

2. Click **Edit** on the toolbar to switch to **Experience Editor**.
3. Click the white area at the top of the page, just below the header, to select the **Two-Column** component.



4. Click the **Remove component** button (the red cross). Most of the content on the page will disappear.
5. Click the grayed area in the middle of the page and click **Add here**.
6. Double-click the **One-Column** component. The content of the page now appears again but in one column.
7. Refresh the browser window to discard the changes.

3. Creating a placeholder settings override for the home page

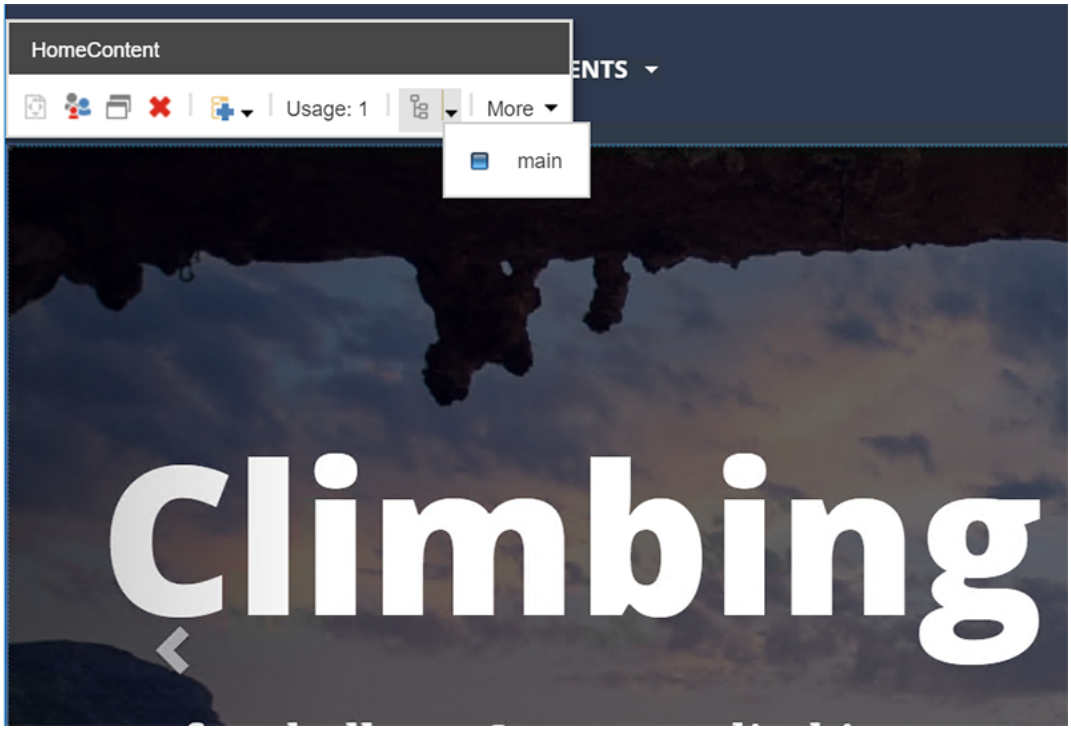
With the current configuration, authors could remove the HomeContent component, which is added to the main placeholder. This should really not be allowed for two reasons: there is no other component that could go in its place currently and the placeholder settings item is configured to only allow the One-Column or Two-Column components on the main placeholder. If authors remove the HomeContent component they won't be able to add it back.

The solution is to create a placeholder override. This means creating a different configuration for the same placeholder for different parts of the site. In this case, the Home page should be treated differently to the rest of the site and not allow adding or removing components from the main placeholder.

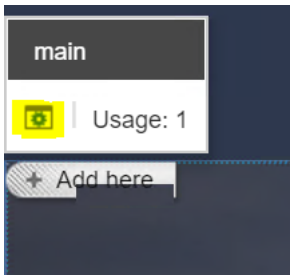
Detailed Steps

1. In the **Experience Editor** open the **Home page**.
2. Ensure the **View > Designing** option is checked.
3. On the **Presentation** tab, click **Final Layout** and select **Shared Layout**. This is to ensure that the changes apply to all versions of the Home page and not just the current one you are viewing.
4. Click the **carrousel** to select the **HomeContent component**.

5. Click the **Component Selector** button from the toolbar to select the main placeholder:

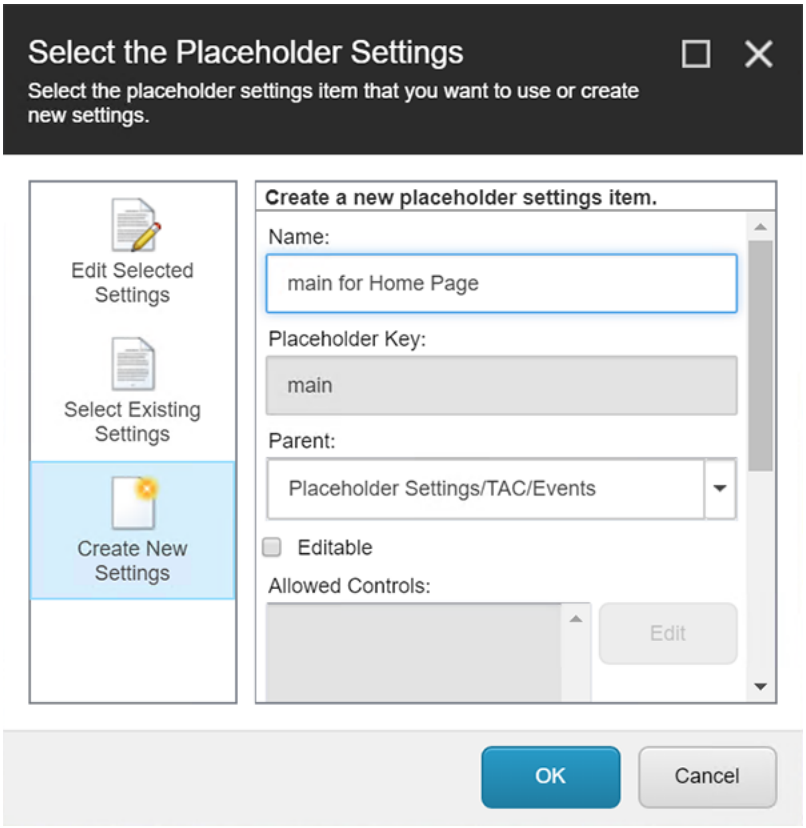


6. Click **Edit Placeholder Settings**.



7. In the **Select Placeholder Settings** dialog, click **Create New Settings**.
8. In the Name text box, type: *main for Home Page*.
9. In the **Parent** field, select the *Placeholder Settings/TAC/Events* folder.

10. Uncheck the **Editable** check box.



11. Click **OK** to close the dialog.
12. Click **Save** on top of the toolbar to save the changes.

Notice that now you can no longer remove the HomeContent component from the home page. However if you go to one of the Event Detail pages, the components inside the main placeholder can still be removed.

4. Componentizing the Home Page further

You are going to componentize the Home page a bit more to allow editors to add and remove components showing the featured events just below the carousel.

Detailed Steps

1. In Visual Studio's Solution Explorer; in the **/Views/TAC/Events** folder, create a new ViewRendering named *FeatureRow*.
2. Remove the
3. Open the **HomeContent.cshtml** file.
4. Locate the [Featured Events] comment.

5. **Cut** the HTML inside the div with the class container and replace it with a placeholder named *features*.

```
<!-- [Featured Events] -->
<div class="container">
    @Html.Sitecore().Placeholder("features")
</div>
<!-- [/Featured Events] -->
```

6. **Paste** the HTML in the **FeatureRow.cshtml** file.
7. **Create** a new ViewRendering named *FeaturedEvent* and remove the empty div.
8. In the **FeatureRow.cshtml** file, copy the inside of the div marked with the col-sm-4 attribute and paste on the FeaturedEvent.cshtml.
9. In the **FeatureRow.cshtml** file, replace the inside of the col-sm-4 divs with three placeholders named: *feature-left*, *feature-middle* and *feature-right*.

```
<div class="row">
    <div class="col-sm-4">
        @Html.Sitecore().Placeholder("feature-left")
    </div>
    <div class="col-sm-4">
        @Html.Sitecore().Placeholder("feature-middle")
    </div>
    <div class="col-sm-4">
        @Html.Sitecore().Placeholder("feature-right")
    </div>
</div>
```

10. **Save** all your changes (press Ctrl+Shift+S)
11. Deploy the Views folder. On the **Solution Explorer**, right-click the Views folder and select **Publish Views.**

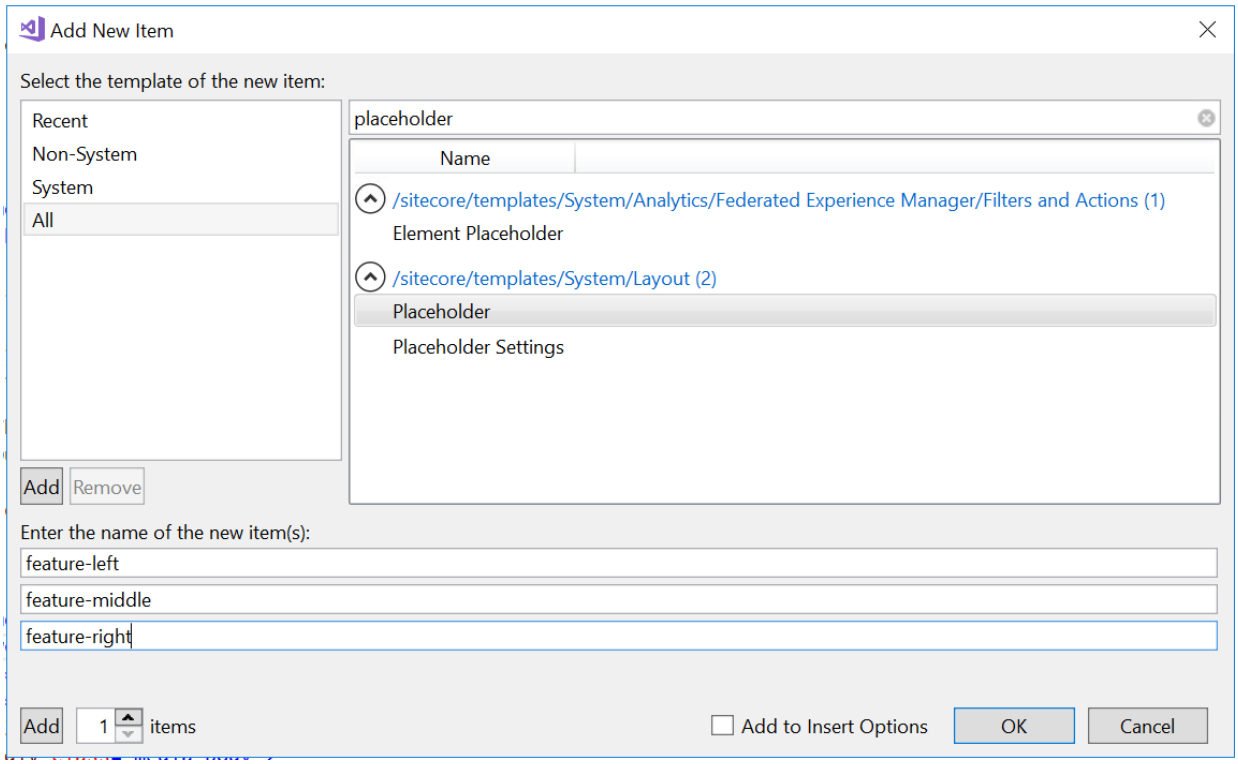
5. Componentizing the Home page further

You now need to create placeholder settings items for the placeholders that you just created. You can leverage Sitecore Rocks bulk item creation functionality.

Detailed Steps

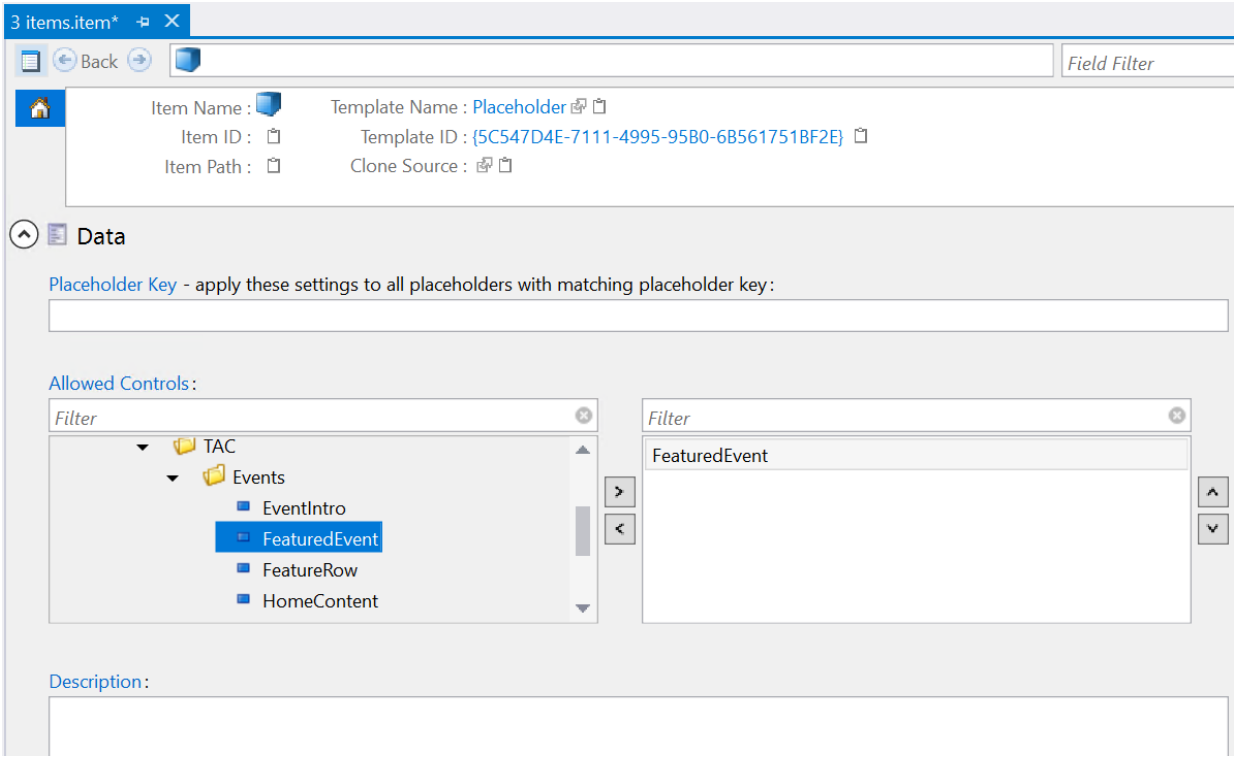
1. In the **Sitecore Explorer**, select the */sitecore/layout/Placeholder Settings/TAC/Events* item.
2. Pres **Alt+Insert** to open the Add New Item dialog.
3. Select **All** and use the filter text to locate the **Placeholder** template.
4. Enter the name *feature*.
5. Click **Add** twice.

6. Rename the items to be **feature-left, feature-middle**, ***feature-right***.



7. Click **OK**

8. In the bulk edit Editing tab, on the title **3 items**tab, set the **Allowed Controls** field to **FeaturedEvent**.



9. **Save** and close the tab.
10. **Create** another Placeholder settings item named *features*, and set its **Allowed Controls** field to **FeatureRow**.

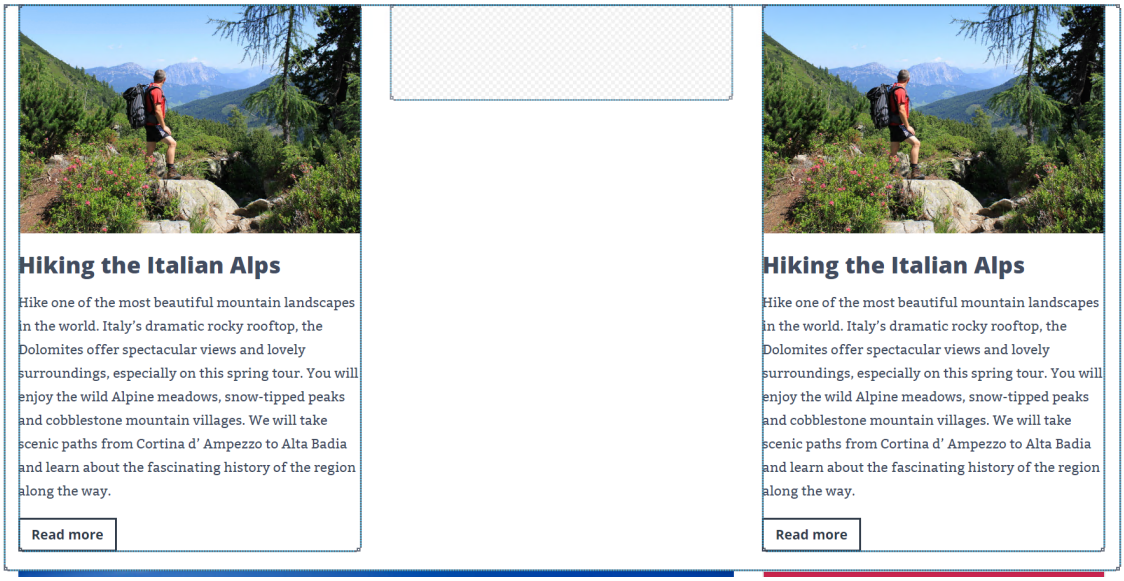
6. Add components onto the Home page

Now you will use the Experience Editor, to compose the presentation of the Home page again, using the newly created renderings.

Detailed Steps

1. Open the **Home page** on the Experience Editor.
2. Ensure that **Designing** is enabled on the **View** tab.
3. In the **Presentation** Tab, **Layout** chunk, make sure **Final Layout** is selected (instead of Shared Layout).
4. Click the **features** placeholder (it is the gray area below the carrousel).
5. Click **Add here.**
6. On the **Select a Rendering** dialog, double-click the **FeatureRow** component. Notice there are now three placeholders instead of one.

7. Using the same method as above, add the **FeaturedEvent** component to the feature-left and feature-right placeholders.



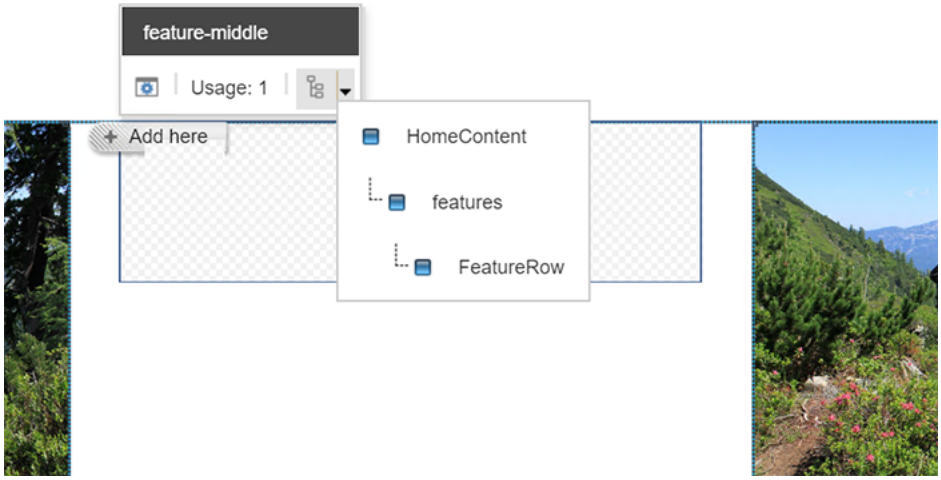
8. **Save** the changes.

7. Adding multiple rows of components

Authors have the flexibility to add multiple copies of the FeatureRow component. This will cause problems because then there will be placeholders on the page with duplicated keys (the ones defined in the FeatureRow component). Content will appear more than once on the page.

Detailed Steps

1. Click the feature-middle empty placeholder and click **Component Selection** to select the **features** placeholder.



2. Click **Add to here** below the Event features.
3. Double-click on the **FeatureRow** component. The component is added, but both rows show the same content.
4. Close the browser to discard the changes.

8. Using Dynamic Placeholders

Detailed Steps

1. In **Visual Studio**, open the **FeatureRow.cshtml** file.
2. Replace all three placeholders for dynamic ones using the *DynamicPlaceholder* method.

```
<div class="row">
    <div class="col-sm-4">
       @Html.Sitecore().DynamicPlaceholder("feature-left")
    </div>
    <div class="col-sm-4">
        @Html.Sitecore().DynamicPlaceholder("feature-middle")
    </div>
    <div class="col-sm-4">
        @Html.Sitecore().DynamicPlaceholder("feature-right")
    </div>
</div>
```

3. **Save** and **deploy** your changes.

4. View the **Home** page in the **Experience Editor**.

   Notice the components inside each feature column are not shown anymore. This is because the the DynamicPlaceholder function uses the given key to create a more complex, unique key. However, the key passed in the method is still used to bind it to the placeholder settings items.

5. **Add FeaturedEvent** components to some of the placeholders and add multiple rows of components as you did in the previous two sections. Notice how now each row can have different components.

MODULE 4: Increasing Component Resusability

In this module, you will learn how to improve the user experience of your customers by allowing users to exchange or reuse components. This makes your site less complex. You will also learn how to use parameters to limit users options and how all components may be cached either statically or dynamically bound.

Lab: Using Data Sources and Setting Data Source Restrictions

In this lab, you will assign a data source to a component so authors can select the item where the data is coming from. You will then set restrictions to ensure authors only select items that are valid for the component.

1. Modify the FeaturedEvent component to display content from Sitecore

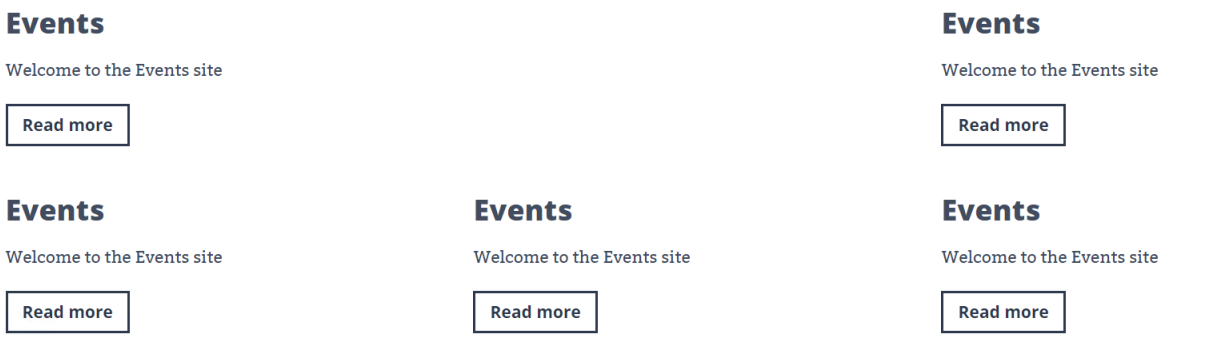
You will start by modifying the component you created in the last module to display content from Sitecore instead.

Detailed Steps

1. Open the **FeaturedEvent.cshtml** file.
2. Replace the static HTML for content from Sitecore using the Field extension method.

```
<div class="thumbnail">
    <a href="Event-Detail.html">
        @Html.Sitecore().Field("EventImage", new {@class="img-responsive", mw=500, mh=333 } )
    </a>
    <div class="caption">
        <h3 class="teaser-heading">@Html.Sitecore().Field("ContentHeading")</h3>
        <p>
            @Html.Sitecore().Field("ContentIntro")
        </p>
        <a href="Event-Detail.html" class="btn btn-default">Read more</a>
    </div>
</div>
```

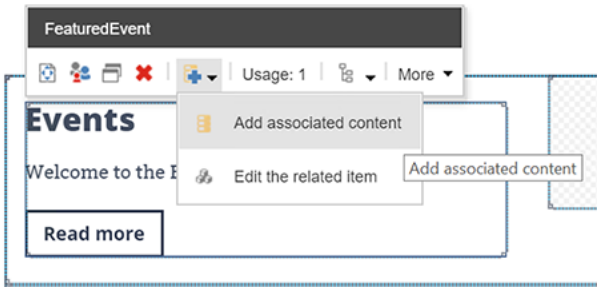
3. Save and deploy the file
4. Preview the Home page of the site. Notice the component is getting the data from the home item. Your Home page may not render any FeaturedEvent components if you didn't save your changes at the end of the last exercise. The following image illustrates 5 FeaturedEvent components saved on the feature-left, feature-middle, and feature-right placeholders- making use of the dynamic placeholders. Add a few more FeaturedEvent components to see the data being retrieved from the Home item.)



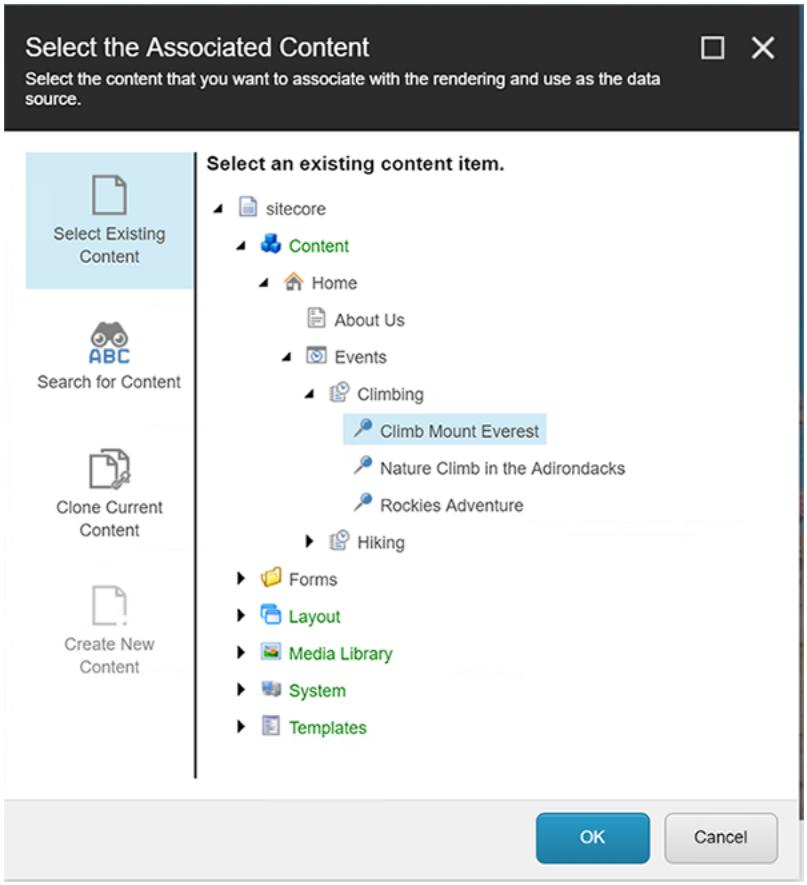
2. Setting a datasource

Detailed Steps

1. Switch to the **Experience Editor** by clicking **Edit** on the **Home** tab of the toolbar.
2. On the **View** tab, ensure that the **Designing** option is checked.
3. Click one of the **FeaturedEvent components** to get the component toolbar.
4. Click **Add associated content** .



5. In the **Select the Associated Content** dialog; navigate and select one of the EventDetail items, for example, **Climbing Mount Everest**.



6. Click **OK**.
7. Repeat the steps to select content for some of the other FeaturedEvent components.

Notice that the content is now coming from the selected item. As long as the **Editing** option on the **View** tab is selected, you should be able to edit the content too.

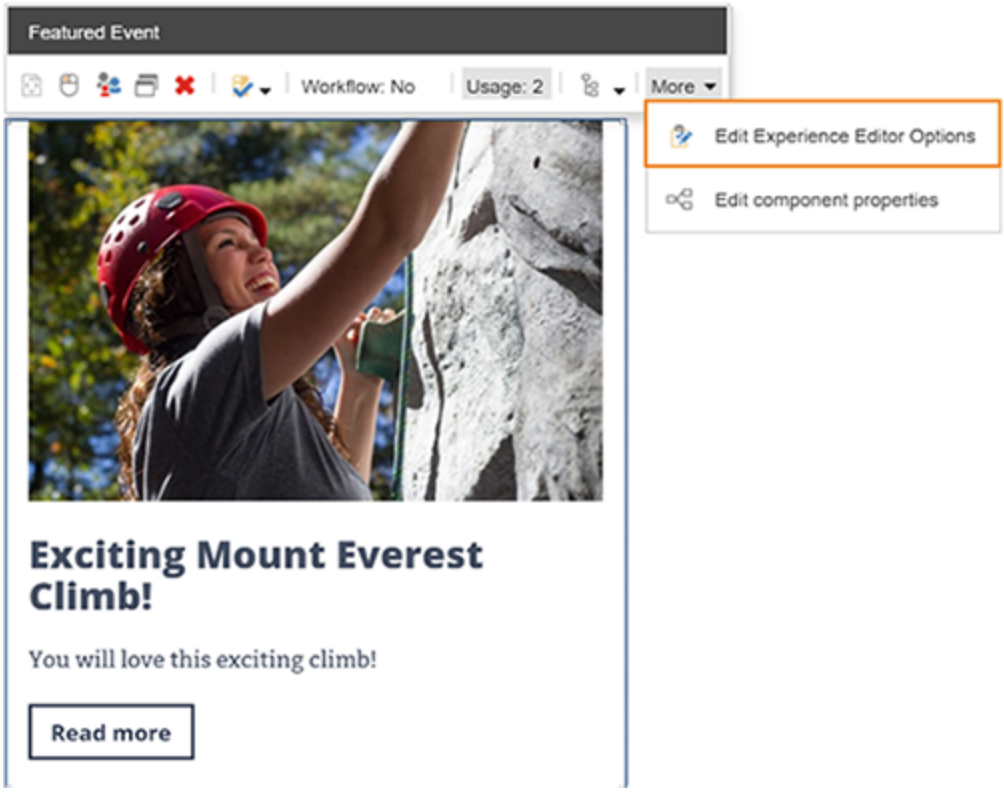
3. Setting datasource template restrictions

To prevent authors from selecting the wrong type of data source for the FeaturedEvent component, you will now implement a restriction to make Event Details the only permissible choice.

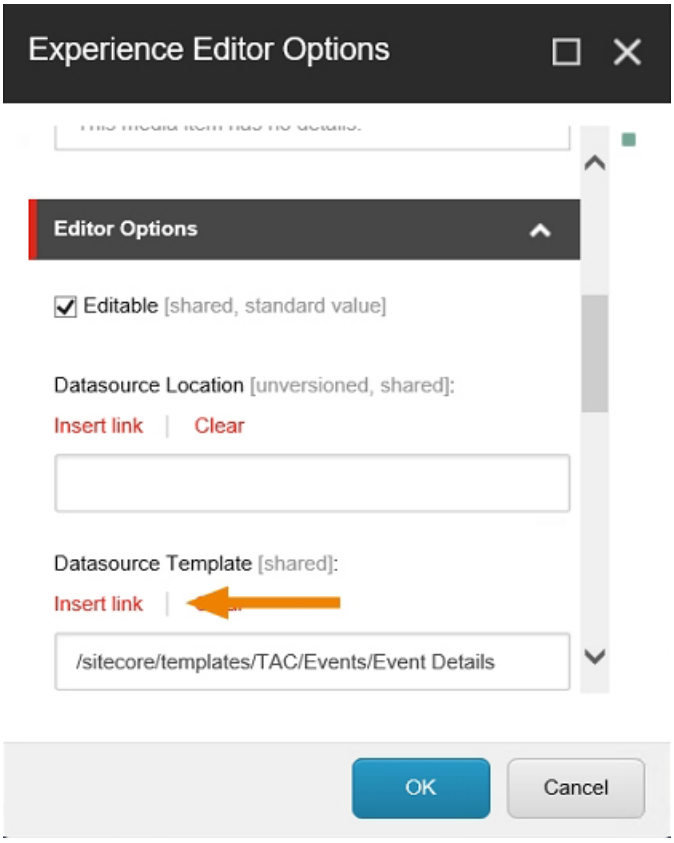
Detailed Steps

1. In the toolbar of the **View** tab, ensure that the **Designing** option is selected.

2. Select one of the **FeaturedEvent** components. In its toolbar, select **More, Edit Experience Editor Options**.



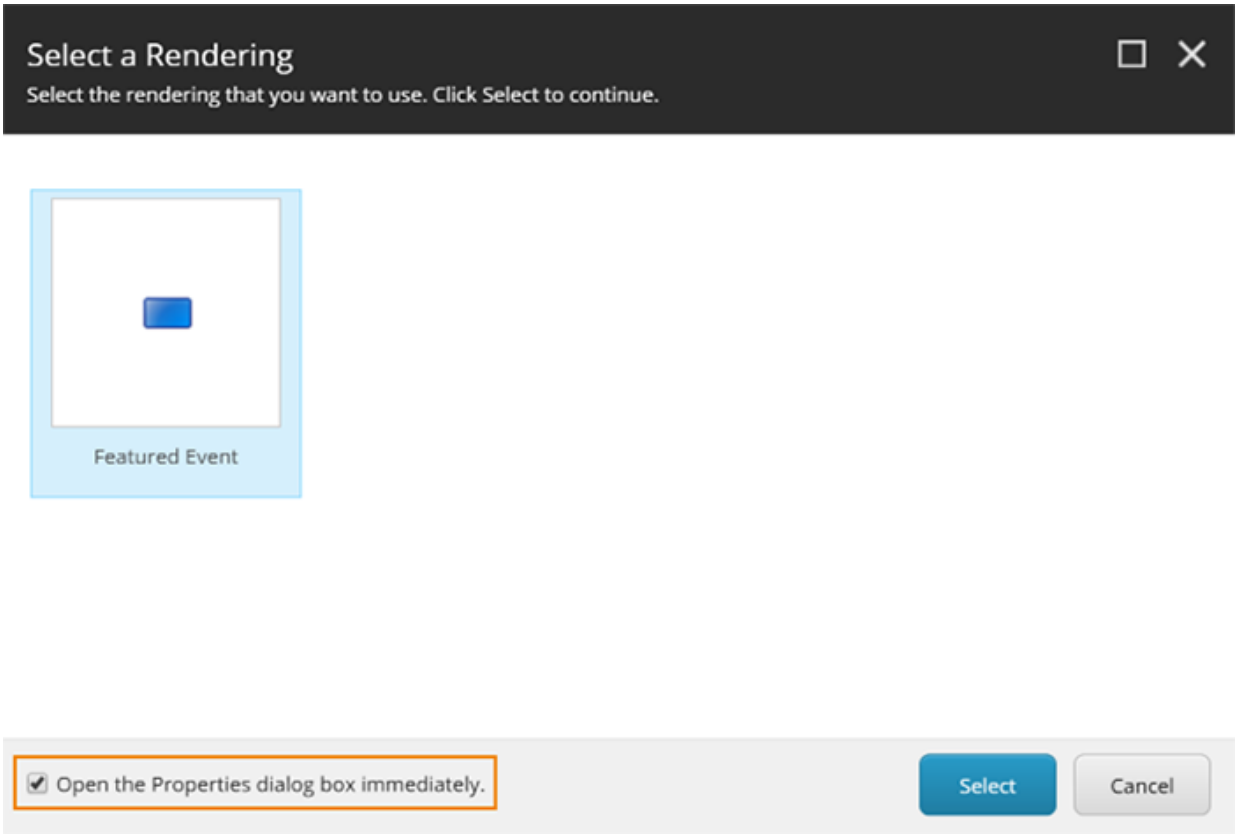
3. In the Editor Options field section, click the **Datasource Template** field's **Insert link** command and navigate to: */templates/TAC/Events/Event Details*. Click **Insert**.



> **NOTE:** Make sure that the Editor Options is expanded, otherwise you can not scroll in this window.

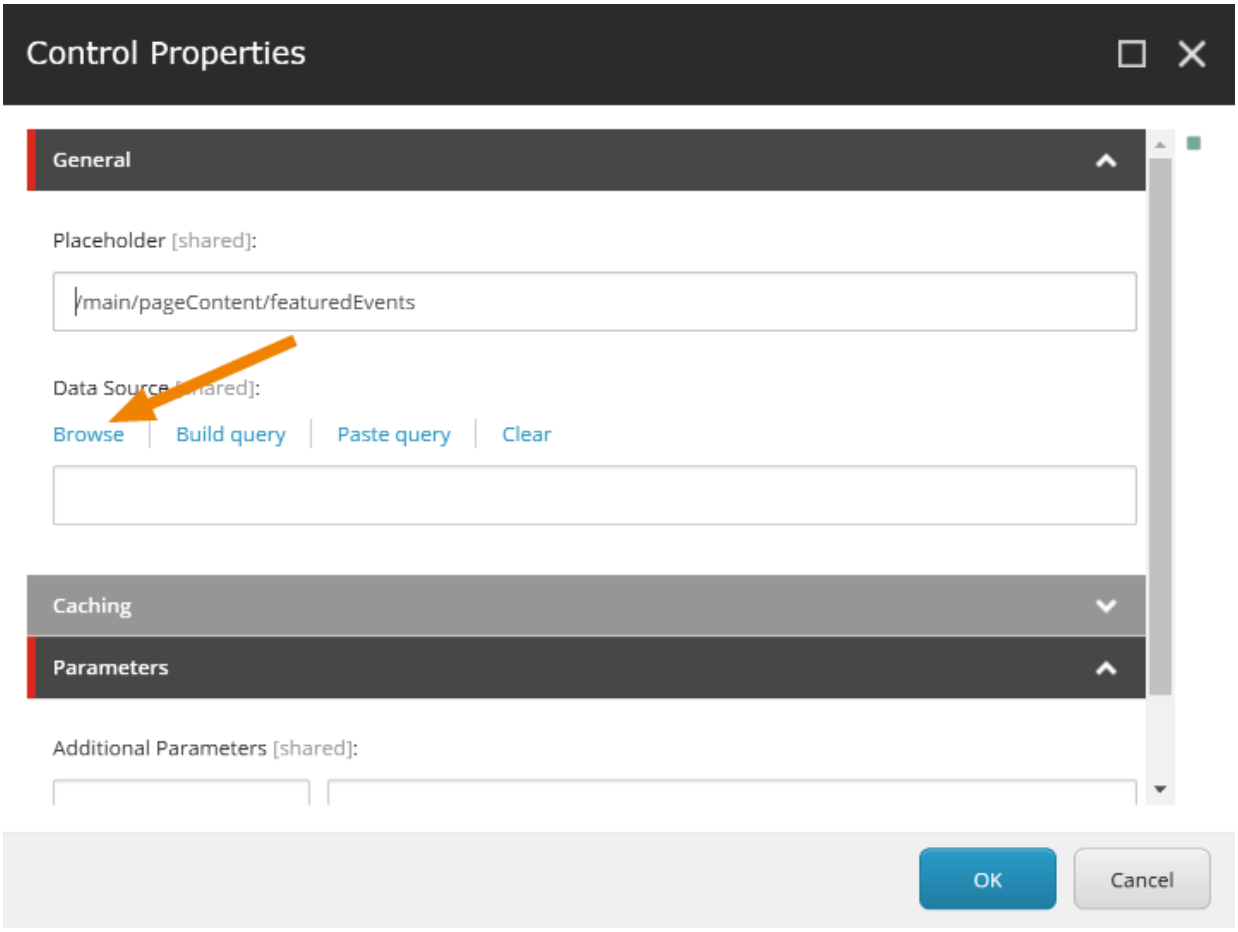
4. Click **OK** to close the **Experience Editor Options** dialog box.
5. Add another FeaturedEvent component to either one of the feature-left, feature-middle, or feature-right placeholders. You can always remove an existing one first if you have added components to all the placeholders.

6. In the **Select a Rendering** window, select **Featured Event** rendering and then select the **Open the Properties dialog box immediately** check box.



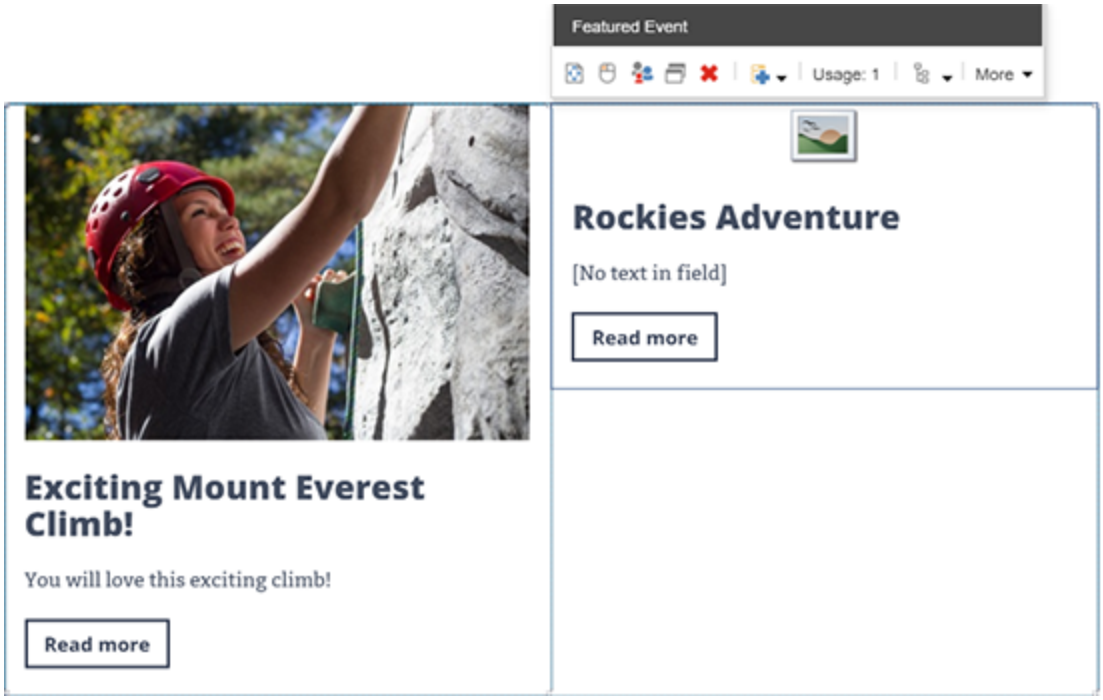
7. Click **Select** and the **Control Properties** dialog box appears.

8. Click **Browse** under the **Data Source** field.



9. Expand the content tree and notice that all items that are not of the type *Event Details* are ghosted (they can't be selected as a valid data source).
10. Select a **ghosted item** to view the invalid selection warning: *The 'sitecore' item is not a valid selection.*

11. Select **Events/Climbing/Rockies Adventure** and then accept your changes.

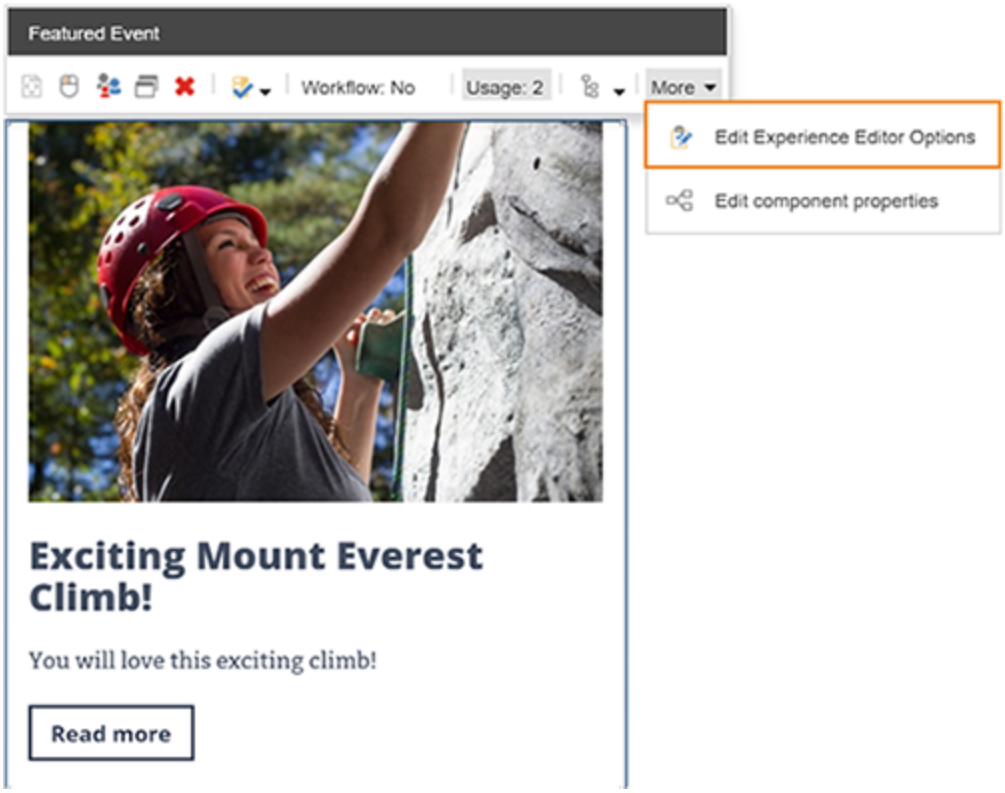


12. Save your changes.

4. Setting a data source location restriction

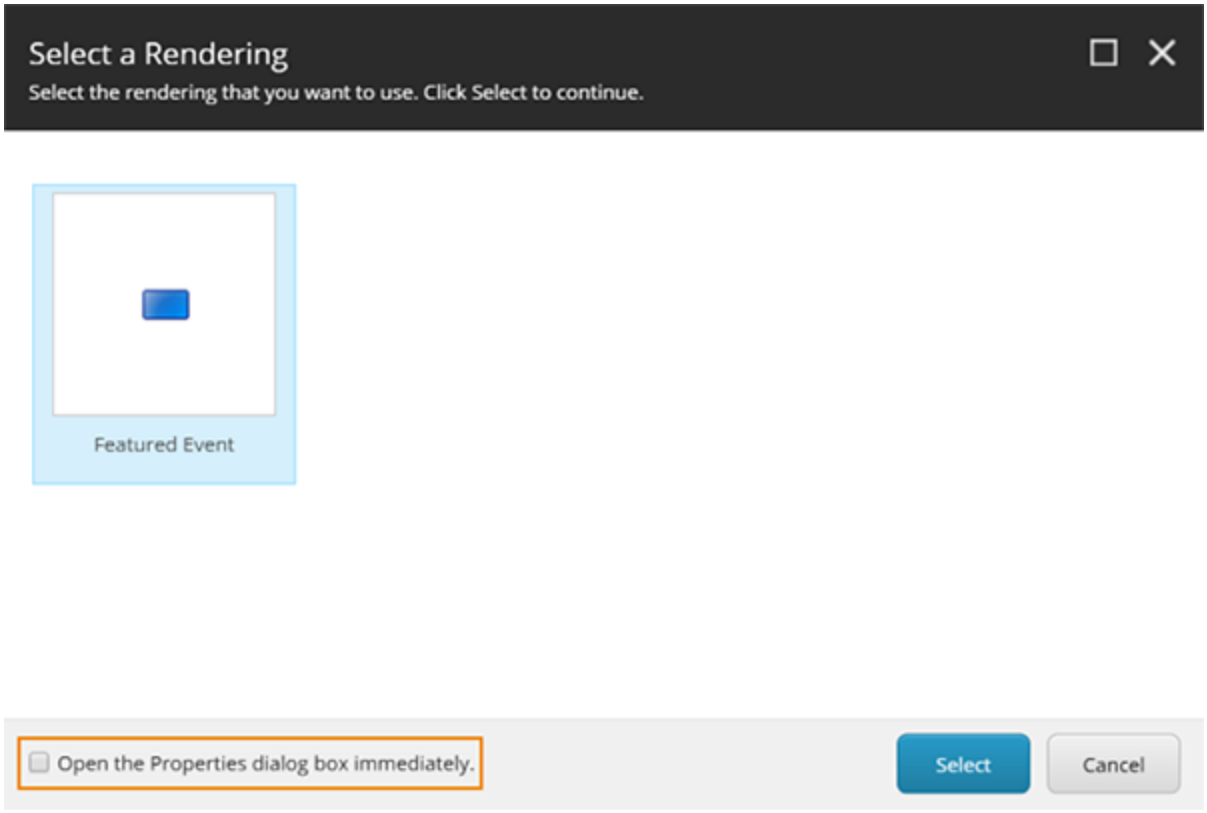
Detailed Steps

1. Select one of the FeaturedEvent components and in its toolbar, select **More, Edit Experience Editor Options**.

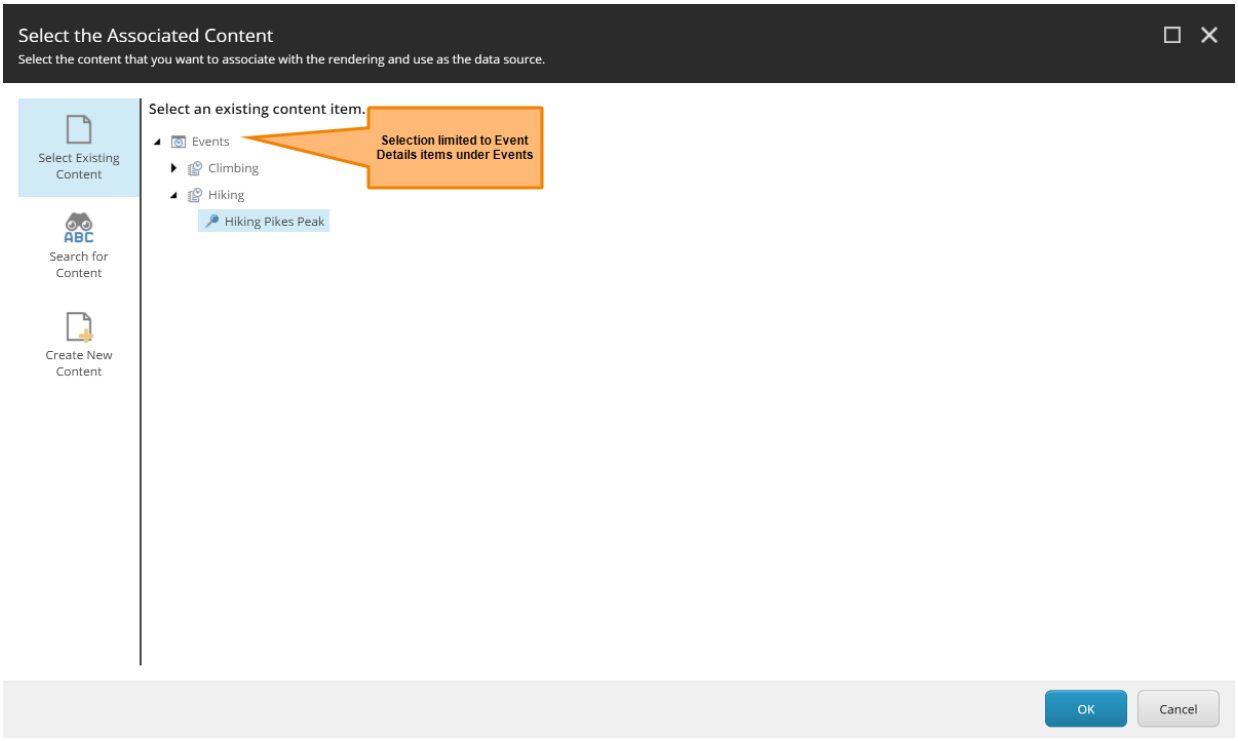


2. In the **Editor Options** field section, click the **Insert link** command for the **Datasource Location** field. Navigate to: *sitecore/Content/Home/Events*. Click **Insert** and confirm your changes.

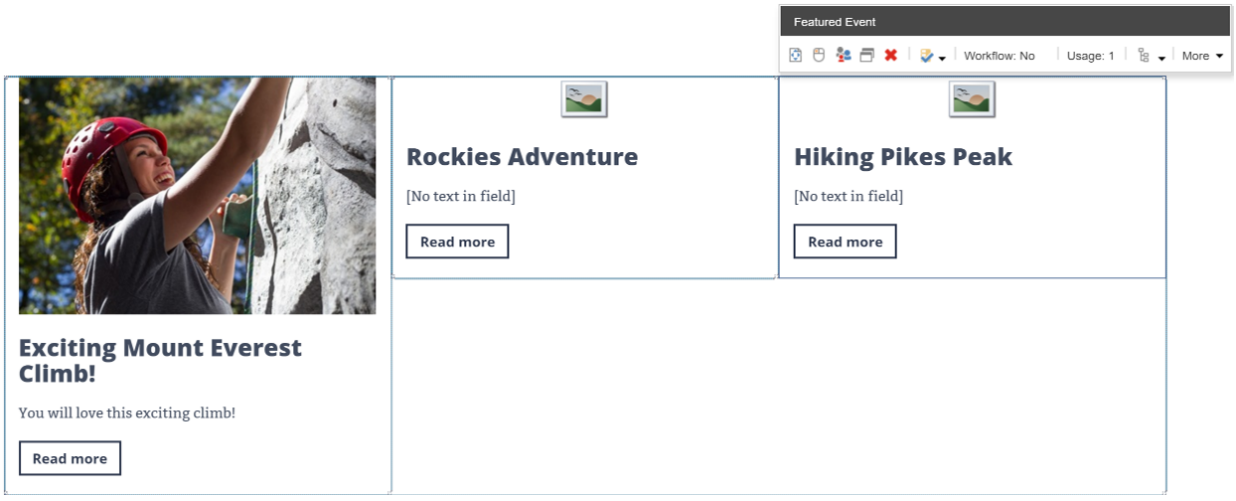
3. In the top-left corner of the toolbar, to the right of the magnifying glass icon, click **Add a new component** and add another FeaturedEvent component to a placeholder of your choice. *Do not* select the *Open the Properties dialog box immediately*. Click **Select**.



4. The **Select the Associated Content** dialog box opens automatically when you set the data source location. Notice the locations that you can select from are limited to the folders under *Events*. The only types of items that you can select are Event Details items.



5. In the dialog, expand the Hiking section and select any **Event Details** item (for example, *Hiking Pikes Peak*). Click **OK**.



6. Save your changes.

STOP HERE

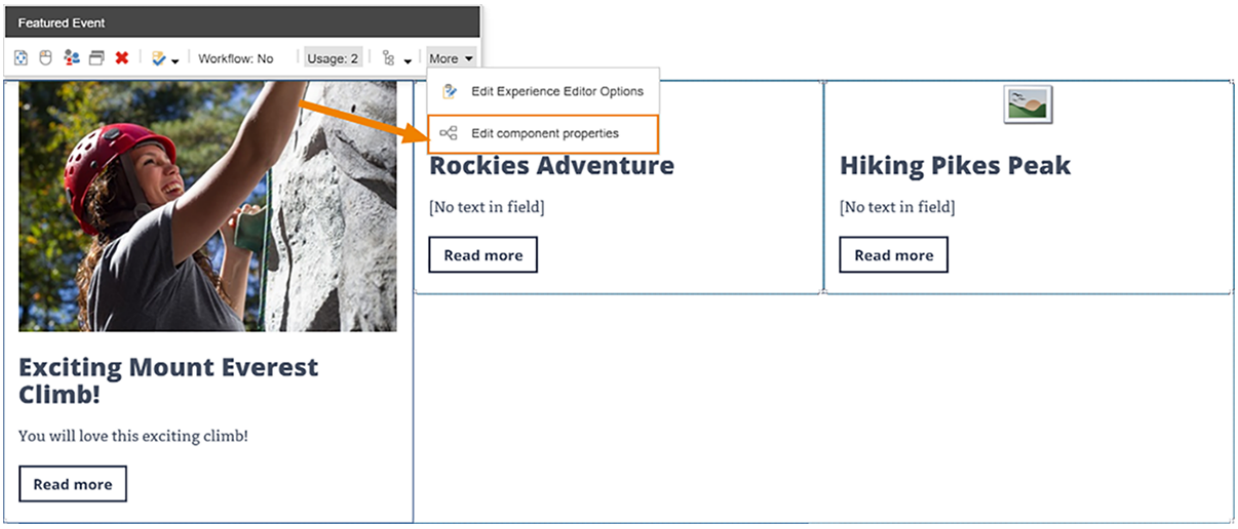
Lab: Working with Component Parameters

In this lab, you will learn how to read extra information from the rendering parameters. This will make components more reusable and will give authors more configuration options.

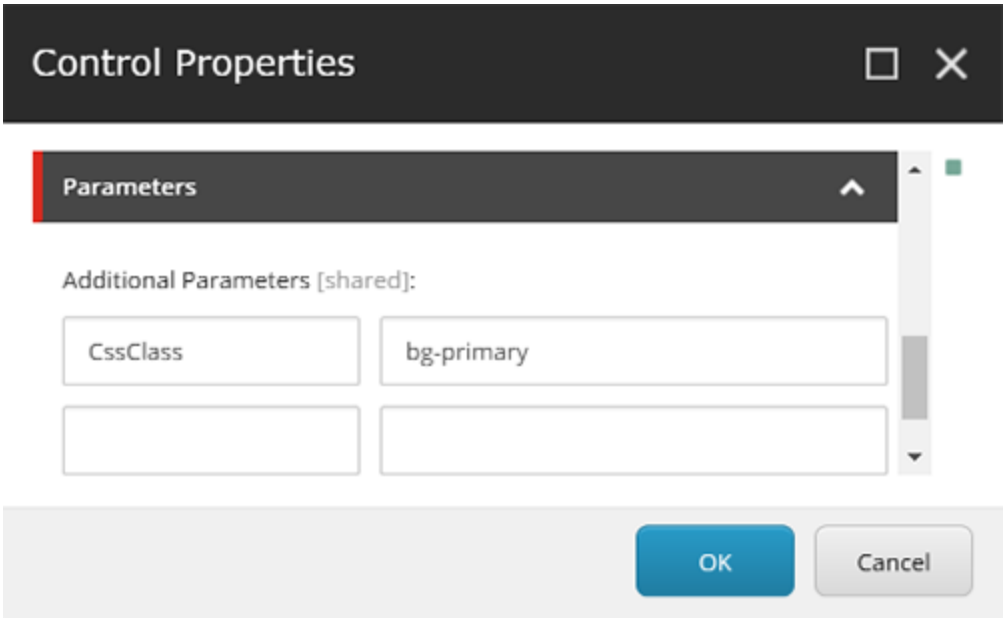
1. Setting component parameters

Detailed Steps

1. In the **Experience Editor**, open the Home item of the Events site in Designing mode, with **Controls** turned on.
2. Select one of the Featured Event components. In its toolbar, select **More, Edit component properties** to open the **Control Properties** dialog box.



3. In the **Control Properties** dialog box, scroll down to the Parameters field section and in the **Additional Parameters** field in the left text box, type: *CssClass*. This will be your parameter name (or key). In the right text box, type: *bg-primary*. This is the value of the parameter.



> **NOTE:** Because the Adventure Company templates are based on Bootstrap, you can use contextual backgrounds to change the background of components. By using contextual backgrounds, you ensure that the look and feel is easily managed. The following contextual backgrounds are available:
>
> - bg-primary
> - bg-success
> - bg-info
> - bg-warning
> - bg-danger

4. Click **OK** to save your changes and close the dialog box, and then click **Save** on the Experience Editor ribbon. Leave the Experience Editor open.

> **NOTE:** You do not see any background color in your component yet. In the next set of steps, you will add code to retrieve and use the CssClass parameter.

2. Modifying the view

Detailed Steps

1. In Solution Explorer of Visual Stuio, open *Views/TAC/Events/FeaturedEvent.cshtml.*

2. Find the ** ** and add a **well** class and retrieve the value of the property from the model:

```
<div class="thumbnail well @Model.Rendering.Parameters["CssClass"] ">
```

<div class="thumbnail well @Model.Rendering.Parameters["CssClass"] ">

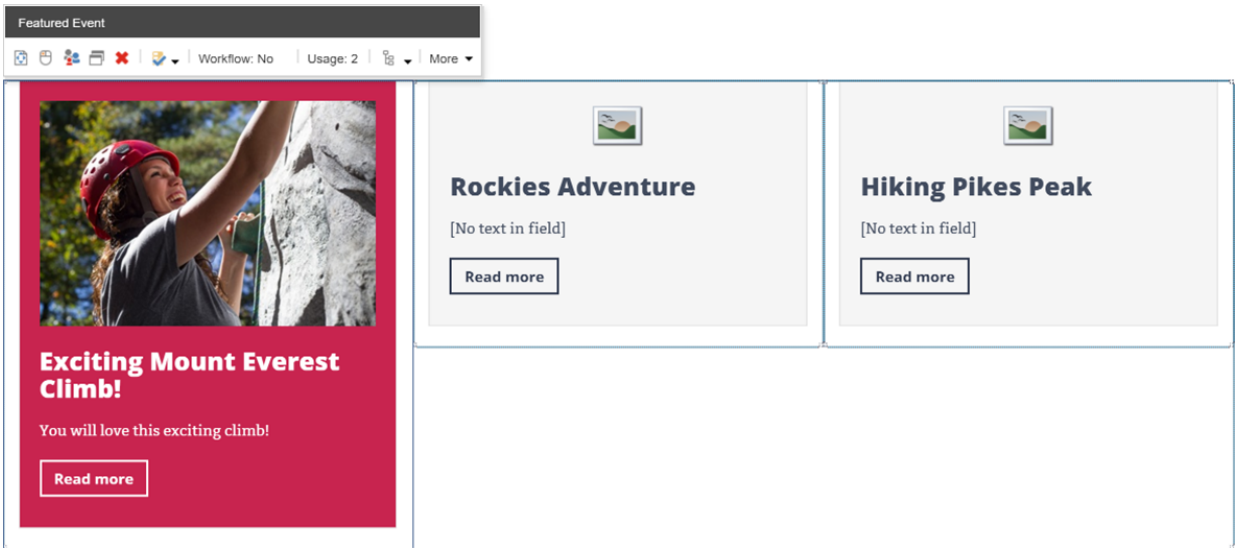
> **NOTE:** The well class adds a rounded border around the FeaturedEvent component with a gray background color and some padding. The grey background color can be altered by adding a contextual background class.

3. Save and deploy the changes to this file.

3. Previewing your changes

Detailed Steps

1. In the **Experience Editor**, refresh the Home page to see the new background color on the Featured Event component.



2. Set the background color on the other FeaturedEvent component properties. You can refer back to the contextual backgrounds to see the available colors.
3. Save your changes.

4. Using parameters with dynamic placeholders

You can use rendering parameters to modify some of the properties of dynamic placeholders. You have to give the property a particular key and the dynamic parameter code will automatically read it and use it.

In general you can override any properties of the dynamic placeholder by passing a rendering parameter of the form *ph_<placeholderkey>_<propertyname>*. Let's assume you have a dynamic placeholder with the key *feature_spot*. The parameter *ph_feature_spot_count* would override the count property. You will now transform the FeatureRow component to take advantage of this functionality and allow authors to vary the number of spots generated.

Detailed Steps

1. Open the **FeatureRow.cshtml** view file.
2. Create a new @functions block and add the following function:

```
@functions{
    TagBuilder CreateDivWrap()
    {
        var tagBuilder = new TagBuilder("div");
        tagBuilder.AddCssClass("col-sm-4");
        return tagBuilder;
    }
}
```

```
@functions{
        TagBuilder CreateDivWrap()
        {
                var tagBuilder = new TagBuilder("div");
                tagBuilder.AddCssClass("col-sm-4");
                return tagBuilder;
        }
}
```

3. Remove all the code inside the div with the row class and replace with a single dynamic placeholder.

```
<div class="row">
      @Html.Sitecore().DynamicPlaceholder("feature_spot", CreateDivWrap(), 3)
</div>
```

4. Save and deploy the file.
5. In the **Sitecore Explorer**, create a new placeholder setting item named *feature_spot* that has only one allowed control: FeaturedEvent.
6. Open the Home page in the **Experience Editor** and refresh the browser. Since you have changed the name of the placeholder they are now empty. Add some FeaturedEvent components to the empty placeholders.

These changes allow you to leverage the dynamic placeholders ability to create a variable number of placeholders. Before, you had one placeholder for each spot; now, you are using the count property, to get three placeholders out of a single dynamic placeholder definition. You had to add the extra function to construct the HTML that should be wrapping each of the placeholders. Now you are ready to overwrite that count property (which is currently hard-coded to 3) with a rendering parameter.

You do have one challenge, you need to vary the width of the spots (the *col-sm-x* div class) according to the number of placeholders to be built.

Detailed Steps

1. Open the **FeatureRow.cshtml** view file.
2. Modify the CreateDivWrap function as follows:

```
@functions{
    TagBuilder CreateDivWrap(DynamicPlaceholderRenderContext context)
    {
        var tagBuilder = new TagBuilder("div");
        int[] widths = new int[] { 0, 12, 6, 4, 3 };
        tagBuilder.AddCssClass("col-sm-" + widths[context.PlaceholdersCount]);
        return tagBuilder;
    }
}
```

```
TagBuilder CreateDivWrap(DynamicPlaceholderRenderContext context)
{
        var tagBuilder = new TagBuilder("div");
        int[] widths = new int[] { 0, 12, 6, 4, 3 };
        tagBuilder.AddCssClass("col-sm-" + widths[context.PlaceholdersCount]);
        return tagBuilder;
}
```

3. Remove the parenthesis from the CreateDivWrap call in the DynamicPlaceholder function.

```
<div class="row">
    @Html.Sitecore().DynamicPlaceholder("feature_spot", CreateDivWrap, 3)
</div>
```

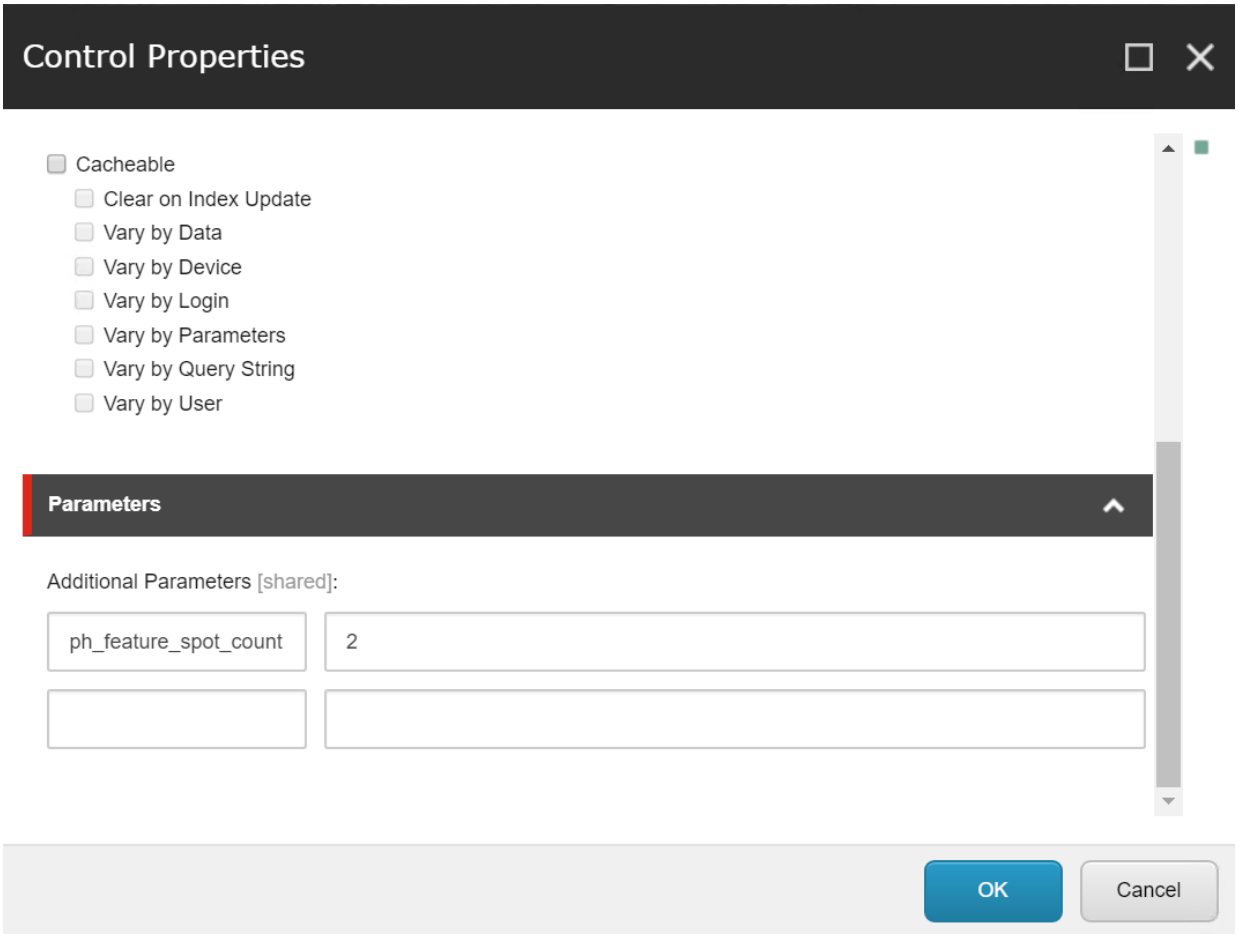
4. **Save** and **deploy** the file.

5. Changing the number of spots in the Experience Editor

Detailed Steps

1. Open the Home page in the **Experience Editor**.
2. Select the FeatureRow component by clicking on the space in between the spots, or selecting a FeaturedEvent component and navigating to it via the *Show ancestors* button.
3. Open the component properties (*More* > *Edit component properties*)

4. Add a parameter with the key *ph_feature_spot_count*, and value 2.



5. Click **OK**.

Notice that the number of spots changes, and the size automatically adjusts accordingly.

6. Setting the maximum number of placeholders

The number of spots on one row should be limited to a maximum. The code you have added is actually not very robust and it will break if the user selects more than 4. You can pass another parameter to the dynamic placeholder function to limit the maximum number of placeholders it will produce:

```
<div class="row">
    @Html.Sitecore().DynamicPlaceholder("feature_spot", CreateDivWrap, 3, 4)
</div>
```

This code will produce 3 placeholders by default, unless overridden through a parameter, and will not allow creating more than 4.

STOP HERE

Lab: Using Rendering Parameters Templates

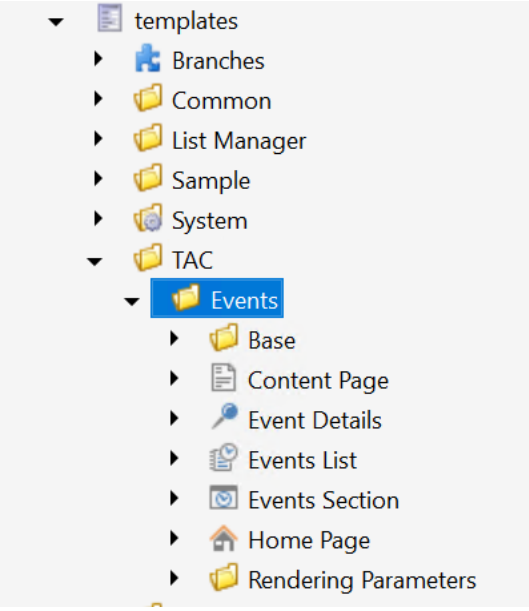
In this lab, you will learn how to create a rendering parameters template to ensure that only valid values are assigned to parameters. This will greatly improve the usability of the rendering parameters feature.

1. Create the rendering parameters template

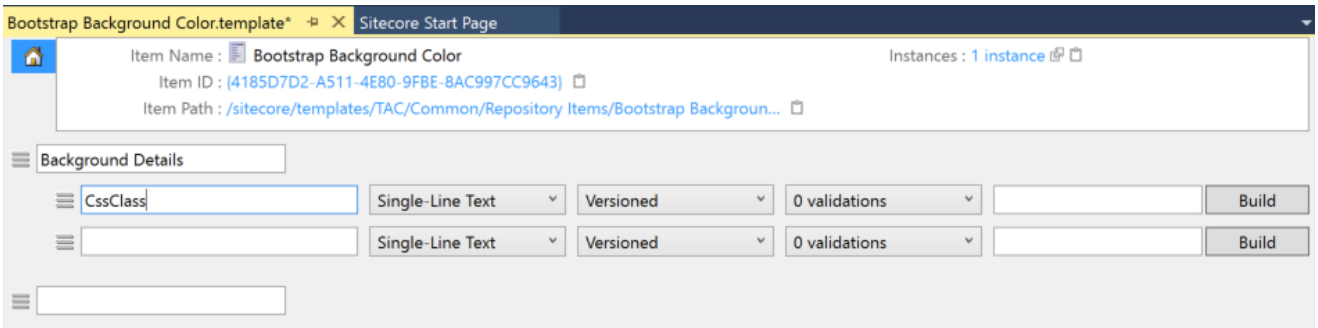
Content authors should only set component parameters if a rendering parameter template is assigned to the component. You will now create the rendering parametesr template.

Detailed Steps

1. In the **Sitecore Explorer**, navigate to */sitecore/templates/TAC/Events/* and create a new Template Folder named *Rendering Parameters*.



2. Right-click the **Rendering Parameters** folder and create a new template named: *Bootstrap Background Color*
3. Add a field section called *Background Details* and then add a field named *CssClass* with the field type of *Single-Line Text*. Save all changes.



NOTE: There is no need to assign an icon to the template because it will not be used to create items.

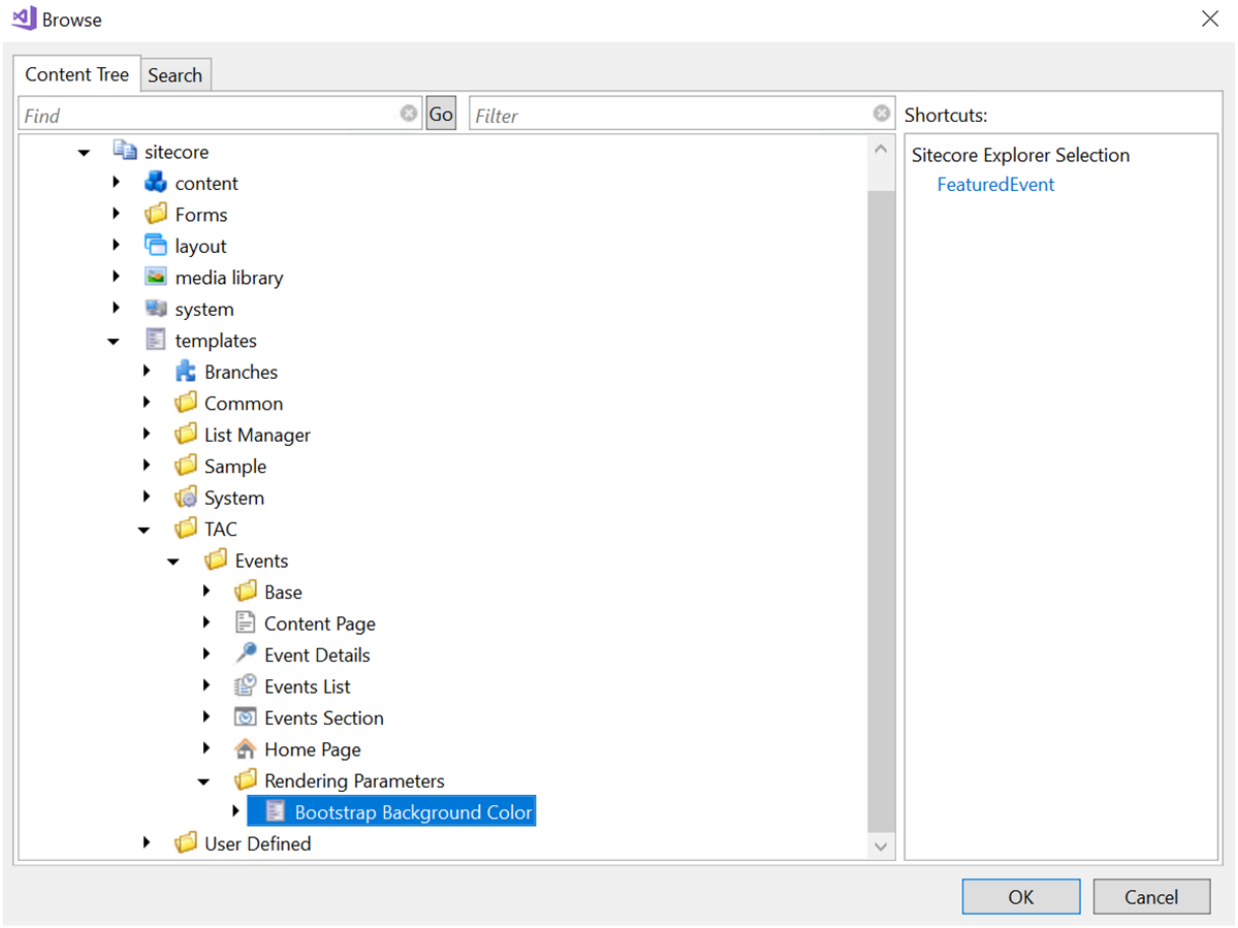
2. Assign the rendering template to the component definition item

To make sure that the component can use the rendering template, the rendering template has to be assigned to the component definition item.

Detailed Steps

1. In the **Sitecore Explorer**, open to the FeaturedEvent component definition item. This is located at: */sitecore/layout/Renderings/TAC/Events/FeaturedEvent.*
2. Scroll down until you find the **Parameters Template** field.

3. Click **Browse** and select the **Bootstrap Background Color**. Save your changes.



4. The parameters template with the field *CssClass* is now assigned.

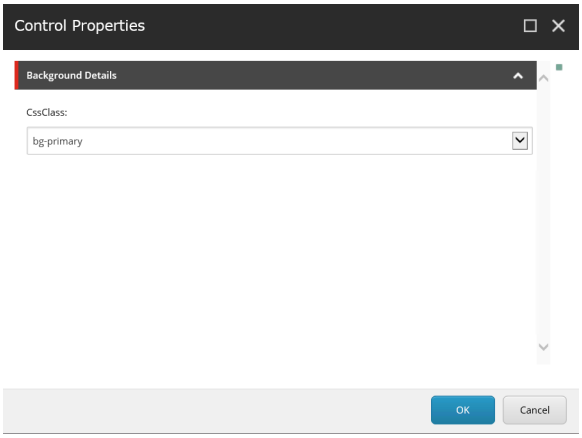
3. Working with the rendering parameters template in the Experience Editor

Now change the background color of one of the featured event components on the Events Home page item. The parameter is now part of the fields in the control properties.

Detailed Steps

1. Open the **Events Home** page in the **Experience Editor**.

2. Click the first **FeaturedEvent component** that appears on the Home page and edit the Background Details in the Control Properties as illustrated below.



The field section and field that you have defined in the rendering parameters template are now part of the Control Properties window. The value has moved because you named the field the same name that you used when you assigned the raw parameter.

4. Manage parameter selections

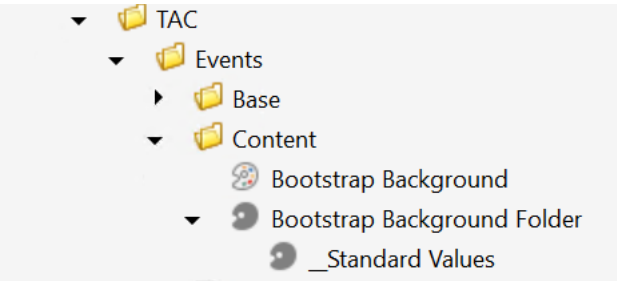
It can be challenging for an author to remember which classes can be used if you have to type them from memory. To help them and avoid human error, you can provide a list of choices, by assigning field types as either a **Droplink** (where a GUID reference is stored), or a **Droplist** (where the name is stored as a string) in the rendering parameters template. When a field source is specified for that field, the drop-down list that is generated for the author is populated with the child items of the referenced item (by either its GUID or path).

You will create a few items to store this information and show them in a dropdown list.

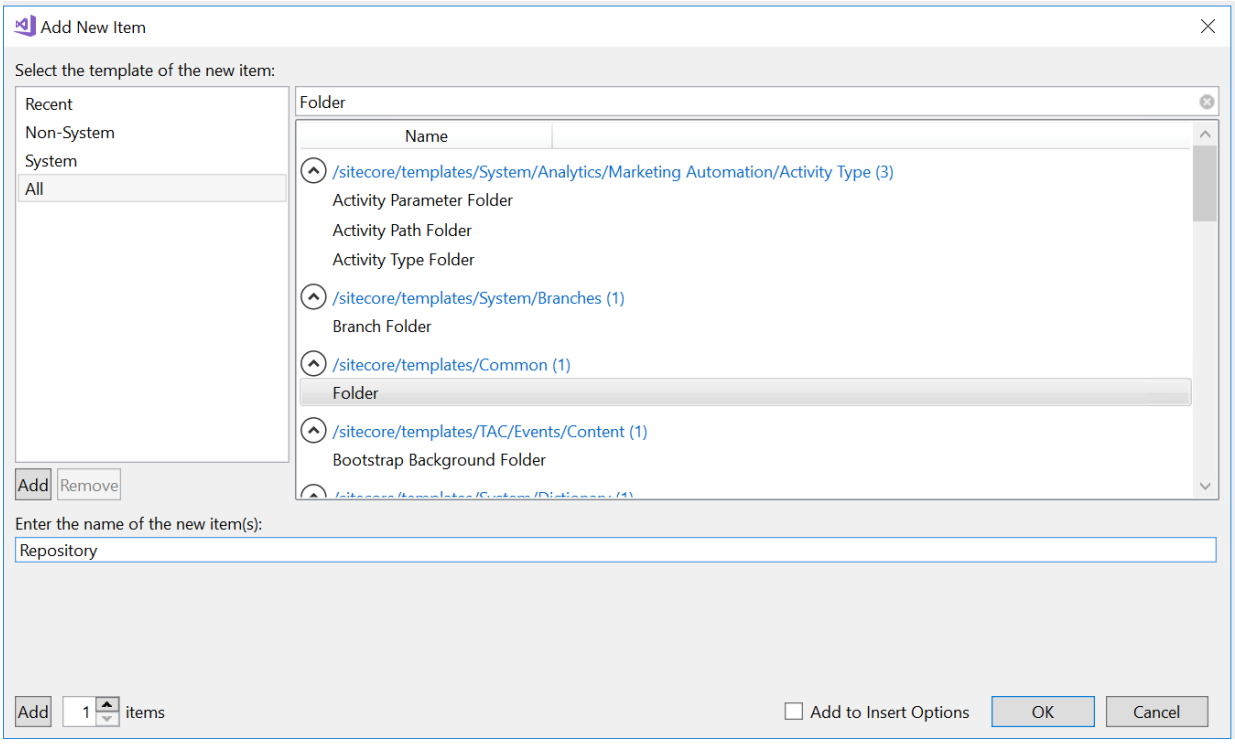
Detailed Steps

1. In the **Sitecore Explorer** in the */templates/TAC/Events* folder, create a new template folder named *Content*.
2. Create a new template named *Bootstrap Background* and set its icon to *Office/16x16/painters_palette_empty.png*.
3. Create a new template named *Bootstrap Background Folder* and set its icon to *Office/16x16/painters_palette.png*.

4. Create the **Standard Values** for the Bootstrap Background Folder and set its insert options to **Bootstrap Background**.

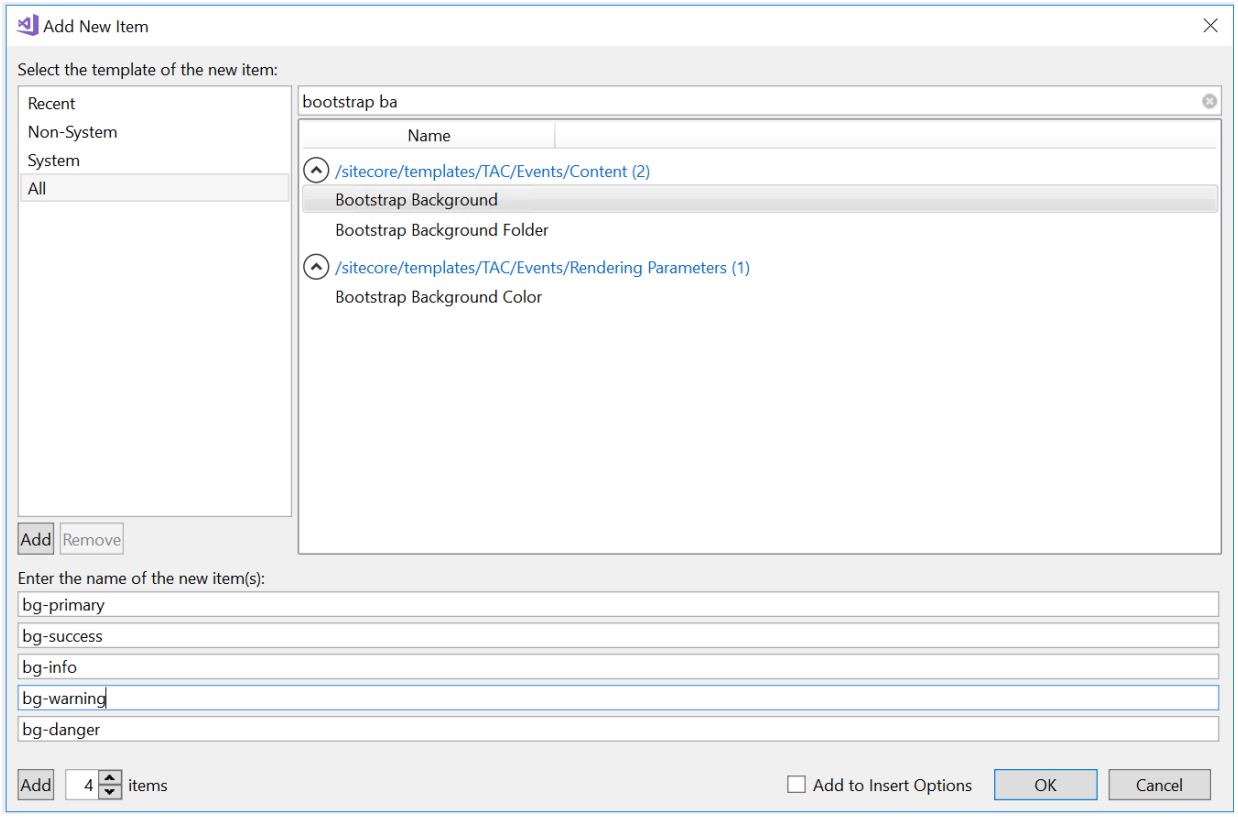


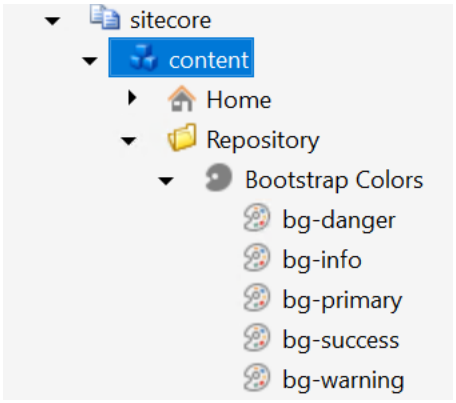
5. Under the /sitecore/content item create a new item based on the folder template named *Repository*.



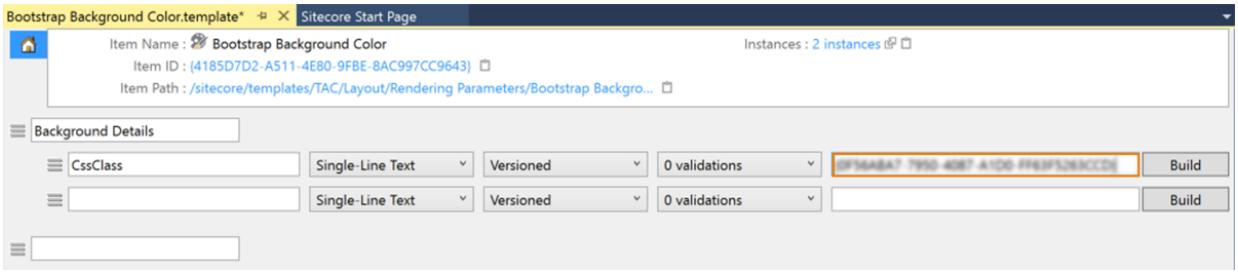
6. Under the **Repository** item, create a new item named *Bootstrap Colors* based on the **Bootstrap Background Folder** template

7. Using the **Bootstrap Background** template create five new items under the Bootstrap Colors item, with the following names:
    - bg-primary
    - bg-success
    - bg-info
    - bg-warning
    - bg-danger





8. Copy the **ID** from the */sitecore/content/Repository/Bootstrap Colors* item (you can use the contextual menu: **Clipboard**, **Copy Item ID**, or the shortcut Ctrl Q+ Ctrl C).
9. Open the **Bootstrap Background Color rendering template** and paste the ID on the CssClass field source.



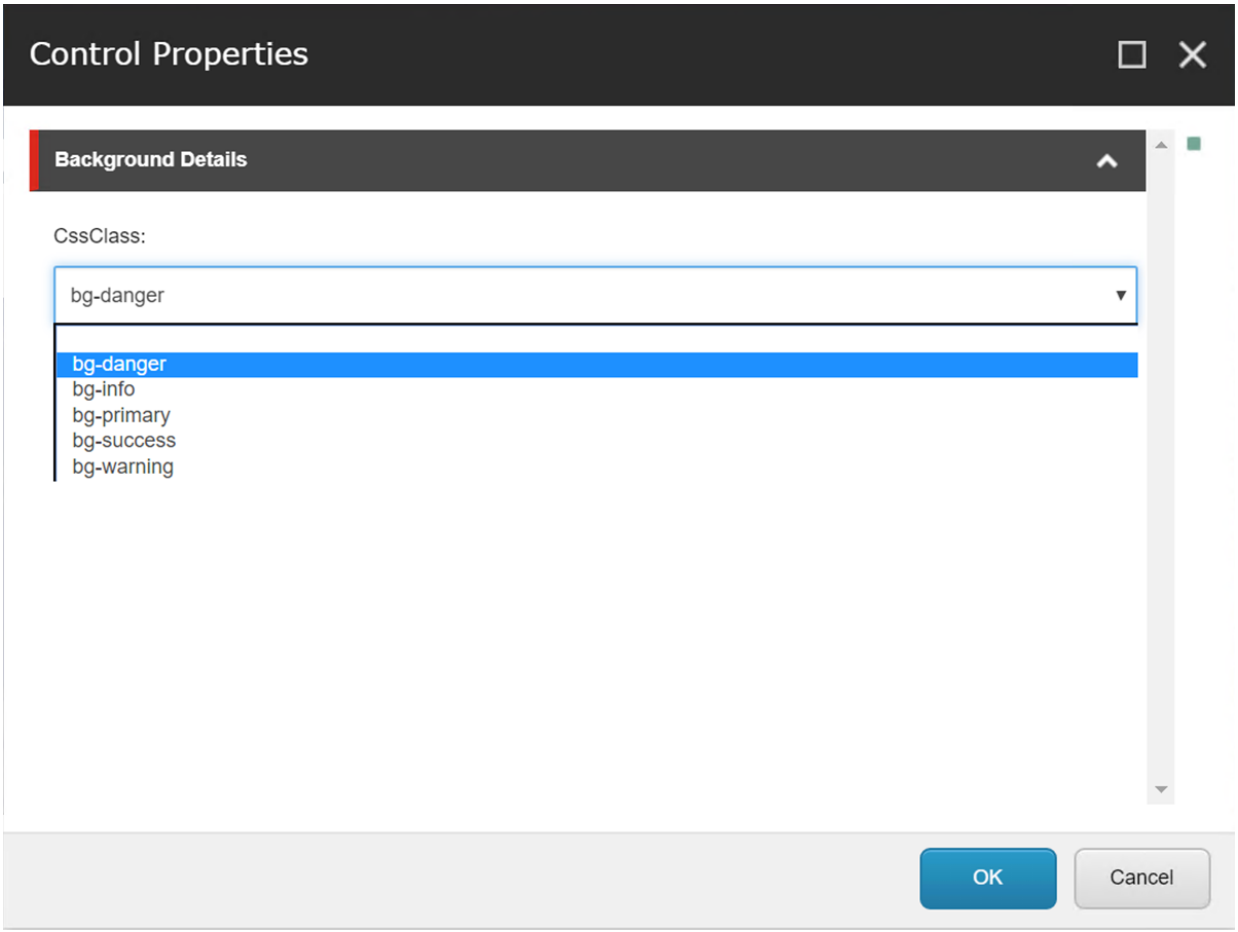
10. Change the type of the **CssClass** field to be **Droplist** and save your changes.

5. Selecting a background from a list

Detailed Steps

1. Open the **Events Home** page in the **Experience Editor**. Notice that the featured event background color is still the same as what you set it to before.

2. Click the first appearing FfeaturedEvent component on the **Home** page and edit the component properties (see below), and click **OK**.



You have now seen how users can assign parameters to a component to set different configuration settings. In this example, you have set a class to change the appearance (the background color) of a component. This makes components highly reusable and it avoids hard-coding content, configuration or other data in components.

6. Adding Rendering parameter template for the FeaturedRow component

Follow a similar approach to create a new template to let authors change the ph_feature_spot_count parameter in the FeaturedRow component.

Detailed Steps

1. Create a new template named *FeatureRow Properties* under the **TAC/Events/Rendering Parameters** template.
2. Add a new section Configuration.

3. Add a new field *ph_feature_spot_count* of type **Single-line text** and save the changes.

   > To provide a nicer looking field name in the UI, you could open the field definition item: */sitecore/templates/TAC/Events/Rendering Parameters/FeatureRow Properties/Configuration/ph_feature_spot_count* and change the Title field.

4. Open the **FeatureRow** component definition item and set the **Parameters Template** field to point to the template you just created.
5. **Save** all your changes.
6. Open the **Home page** in the Experience Editor.
7. Navigate to a **FeatureRow component** and open the **Control Properties.**
8. Inspect the new field and modify its value.

You could convert the single line text field to a droplist to make it even easier for authors.

STOP HERE

Lab: Working with Compatible Renderings

In this lab, you will configure various renderings as compatible so authors can easily switch between them.

1. Configuring compatible renderings

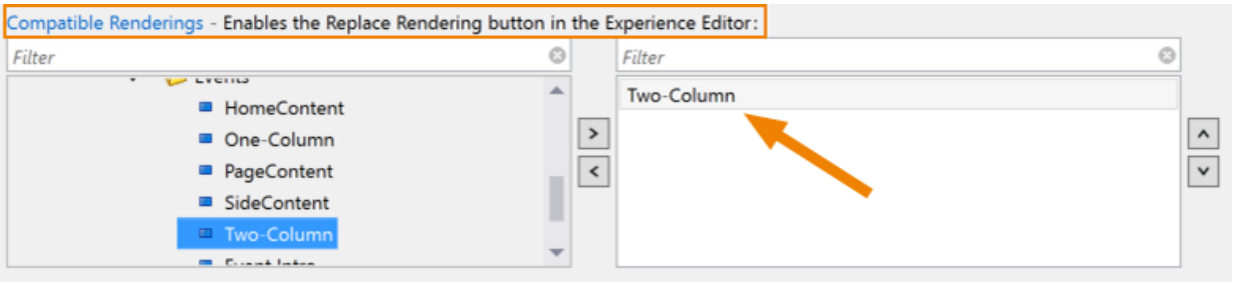
Certain components, such as One-Column and Two-Column components, are interchangeable. This means that they:

- Perform a very similar function, but may have a slightly different design.
- Accept the same data source. This is not a requirement, but compatibility suggests that content does not need to change when the component data source changes.
- Accept the same rendering parameters. This is not a requirement, but compatibility suggests that the rendering parameter selections are also compatible.

Sitecore enables you to configure a list of components that are compatible with the currently selected component. You can do this by using the Compatible Renderings multilist on the component definition item:

Detailed Steps

1. In Sitecore Explorer, double-click the **One-Column Component Definition** item found at: */sitecore/layout/Renderings/TAC/Events/One-Column*.
2. On the **Editor** tab, scroll to the **Editor Options** field section. In the **Compatible Renderings** field expand the layout tree and double-click to select the **Two-Column** component found at: */layout/Renderings/TAC/Events/Two-Column*

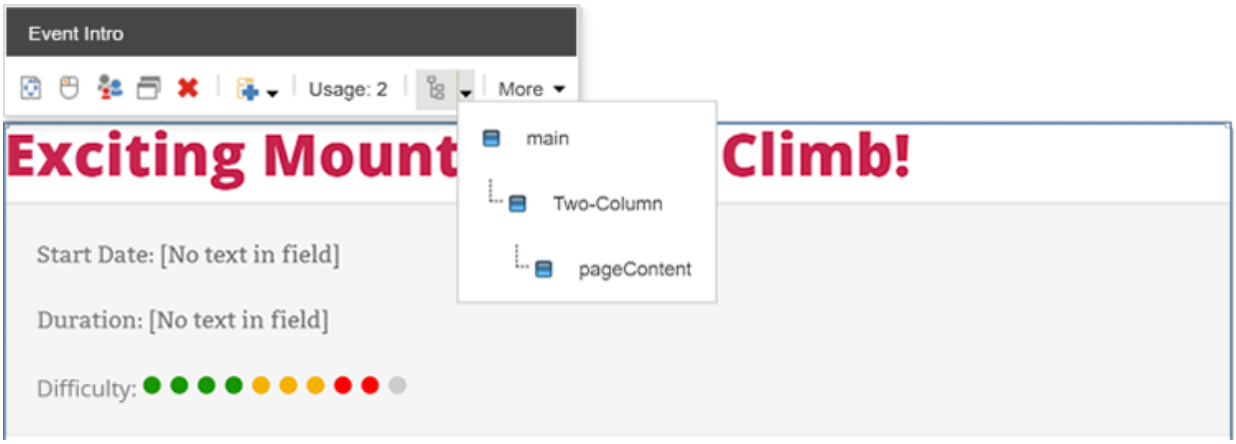


3. Save your changes and close the Editor tab.
4. Open the **Two-Columncomponent definition** item and add **One-Column** to its **Compatible Renderings** field.
5. Save your changes and close the Editor tab.

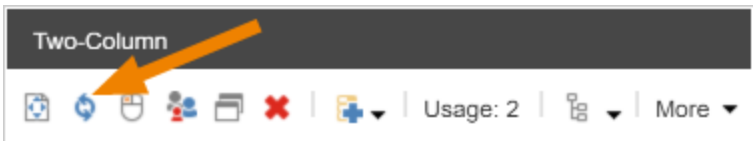
2. Replacing one component with another

Detailed Steps

1. In the **Experience Editor**, navigate to one of the Event details pages (for example: /*site-core/content/Events/Home/Events/Climbing/Climb Mount Everest*)
2. Ensure that **Designing** mode is selected. Select the existing **EventIntro** component and then click the drop arrow (**Show ancestors**) on the parent command to view the component and placeholder hierarchy.

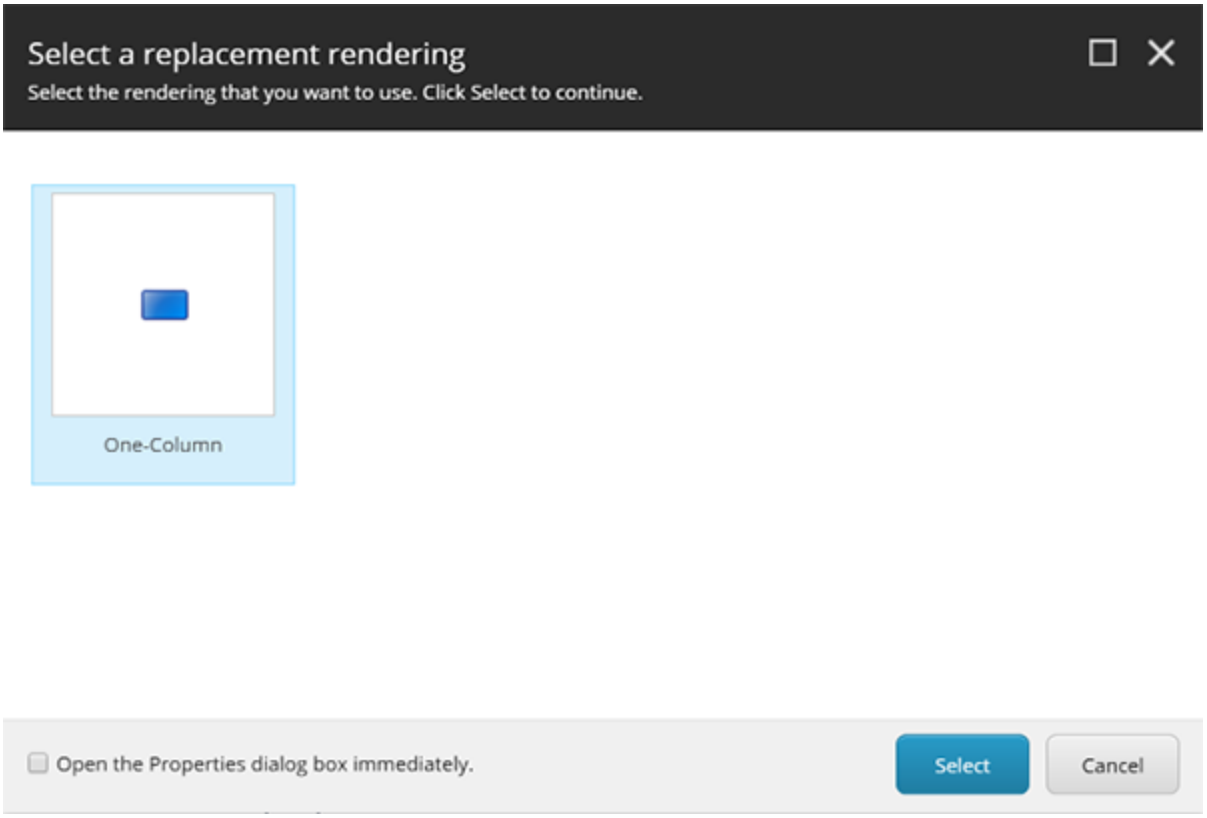


3. In the component hierarchy, select **Two Columns**. Click **Replace with another component**.

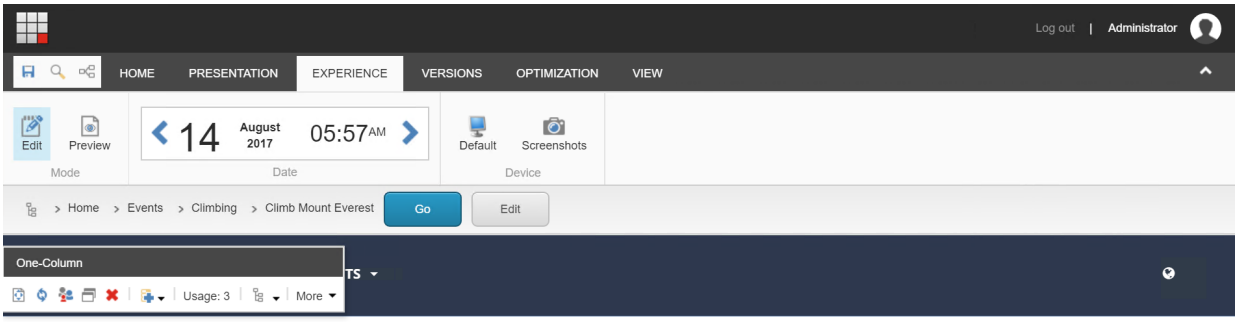


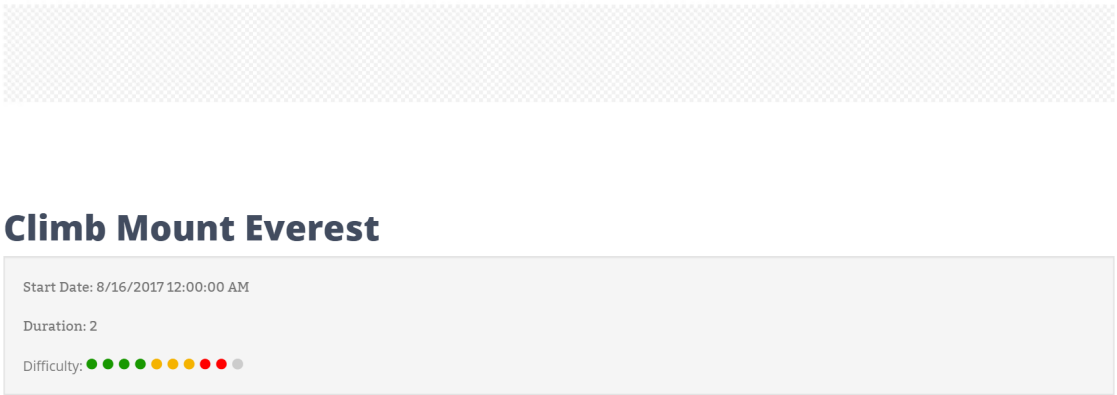
> The *Replace with another component* command is only available when a component definition item has compatible renderings assigned. The *Replace with another component* command is grayed out if the selected component has compatible renderings assigned, but the compatible rendering is *not allowed* in the current placeholder.

4. Select the **One-Column** component as replacement rendering and confirm your changes.

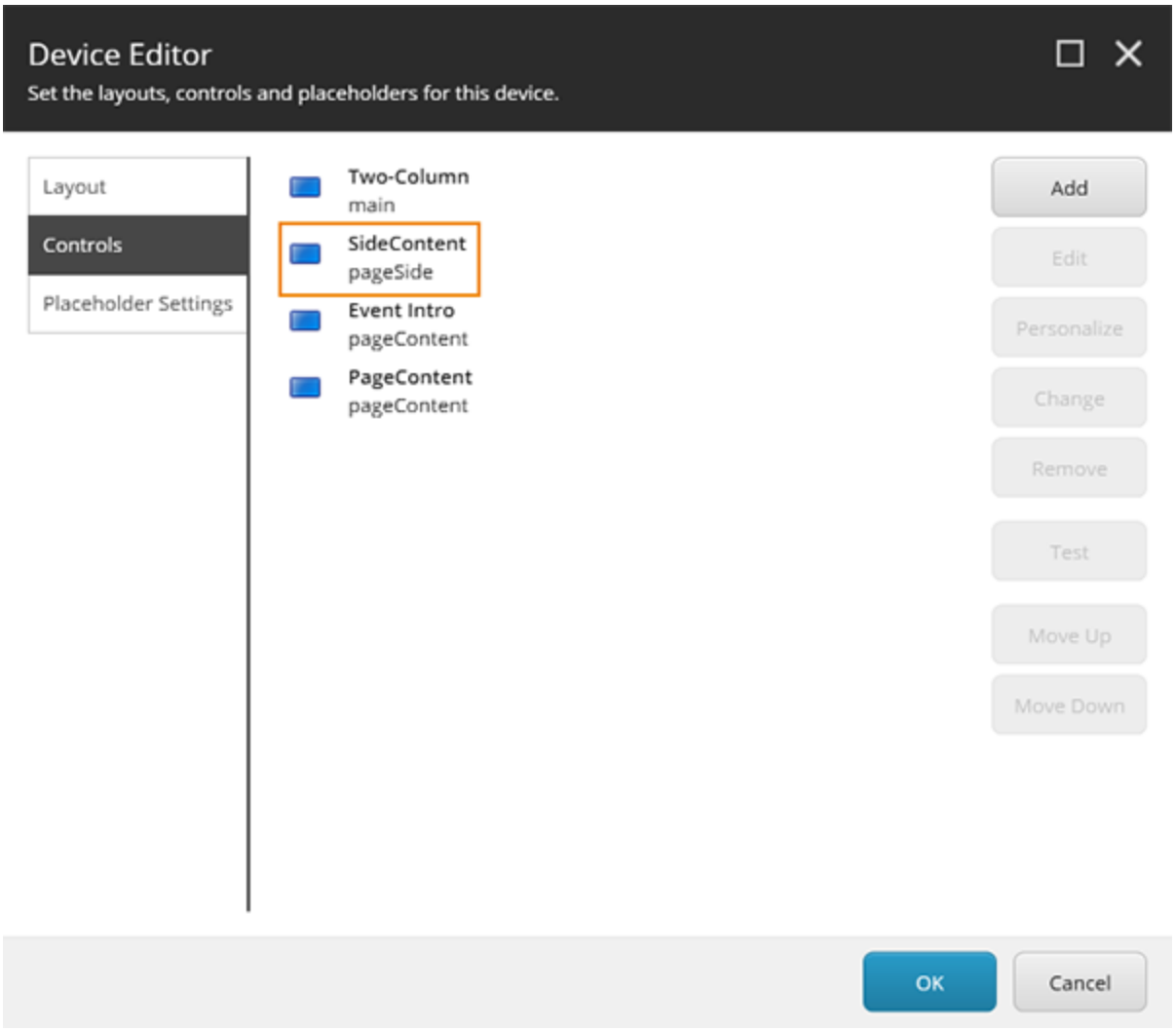


5. Notice that the Event intro and the page content components now use the full page.





6. On the **Presentation** tab, click **Details** to open the **Layout Details** dialog box. Notice that the SideContent component is still part of this page layout details. Click **Edit** for the Default device to open the Device Editor, and then click the **Controls** tab to see which placeholder the component is assigned to. Click **OK** or **Cancel**.



NOTE: The SideContent component is no longer rendered to the browser because the pageSide placeholder is not part of the One-Column layout code file.

7. Replace the One-Column component with the Two-Column component using the **Replace with another component** command. Save your changes.

STOP HERE

MODULE 5: Applying Navigation Practices Within the Site

In this module, you will learn how Sitecore manages URLs and Links. You will learn about the parts of the API that are responsible for resolving URLs to Items and Items to URLs.

Lab: Outputting Links with the LinkManager

In this lab, you will invoke the LinkManager to render links on the page.

1. To edit the code for the View:

Detailed Steps

1. Open the **FeaturedEvent.cshtml** under the *Views/TAC/Events* folder.
2. Find the two lines with *<a> tag*,and replace the hard-coded value from the **href property** with the item's URL obtained with the **@Sitecore.Links.LinkManager.GetItemUrl** method. You can get a reference to the data source item using **RenderingModel.Item.**

```
@model RenderingModel
@{
    var url = Sitecore.Links.LinkManager.GetItemUrl(Model.Item);
}
<div class="thumbnail well @Model.Rendering.Parameters["CssClass"] ">
    <a href="@url">
        @Html.Sitecore().Field("EventImage", new {@class="img-responsive", mw=500, mh=333 } )
    </a>
    <div class="caption">
        <h3 class="teaser-heading">@Html.Sitecore().Field("ContentHeading")</h3>
        <p>
            @Html.Sitecore().Field("ContentIntro")
        </p>
        <a href="@url" class="btn btn-default">Read more</a>
    </div>
</div>
```

3. Save and deploy the file.

2. To verify your changes:

Detailed Steps

1. Preview the **Home page**.
2. Hover over a **FeaturedEvent component** and view the URL displayed in the bottom-left of your browser.

STOP HERE

Lab: Creating a Breadcrumb Component

In this lab, you will create a Breadcrumb component that will show links to each item in the navigation path to the current page.

1. Prepare the solution for using unit tests

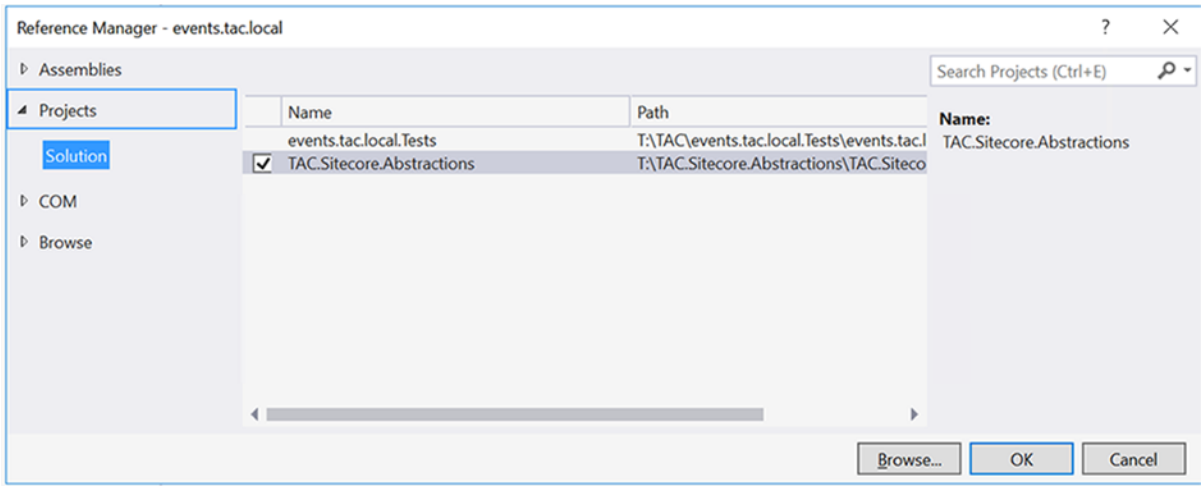
Unit testing with Sitecore can be challenging. This is because there is a lack of interfaces (there is no *IItem*, or *IRenderingContext* provided by Sitecore) and an abundance of static objects (*Sitecore.Context*, *LinkManager*, etc). Newer parts of the Sitecore API provide better support for testing and with every version of Sitecore it gets easier.

Over the years, the community has come up with solutions to these challenges. There are two main approaches: creating an interface layer between your code and Sitecore, or provide a way of faking the Sitecore database. In this elearning, you are going to try the first approach. You will add a small library to your solution that allows you to work against an interface instead of Sitecore objects.

The library provides two implementations of the interface: one that works with Sitecore, and one to use in tests.

Detailed Steps

1. From the Student Resources folder; copy the **TAC.Sitecore.Abstractions** folder to the Visual Studio solution folder.
2. In the **Solution Explorer**, click the solution node and select **Add, Existing Project...**
3. Choose the **TAC.Sitecore.Abstractions.csproj** from the solution root.
4. Add a reference to this new project to the other projects in the solution. Right-click on the events.tac.local project and choose **Add, Reference**.
5. Click **Projects** and select the **TAC.Sitecore.Abstractions project** check box.



6. Click **OK** to close the dialog.
7. Repeat the steps to add a reference to TAC.Sitecore.Abstractions to the test project.

8. In the **events.tac.local.Tests project,** delete the **Test1.cs class**.
9. Right-click the **events.taclocal.Tests project** and select **Add, Existing item...**
10. Select the **BreadcrumbBuilderTests.cs** from the **Student Resources** folder.
11. Right-click the **events.taclocal.Tests project** and select ,once more, **Add, Existing item...**
12. Select the **TestTree.cs** from the **Student Resources** folder.

2. Creating the model

You will start by creating the model definition, which is just a POCO class with a few properties to store the data required by the breadcrumb.

Detailed Steps

1. Create a new class **NavigationItem** under the **Models** folder of the events.tac.local project.
2. Add the following properties to the class with a *public get* and a *private set*:

Property Type	Property Name
string	Title
string	URL
bool	Active

3. Add a **public constructor** to initialize those properties. Provide a default value of **false** for the active parameter. Your **NavigationItem** class should look like this:

```
public class NavigationItem
{
    public NavigationItem(string title, string url, bool active = false)
    {
        Title = title;
        URL = url;
        Active = active;
    }

    public string Title { get; private set; }

    public string URL { get; private set; }

    public bool Active { get; private set; }
}
```

3. Creating the business layer

The recommended practice is to have very simple controllers, so you will put all the logic building the Breadcrumb component in a separate class in the business layer. This class will create the model that then will be passed to the view, via the controller. This is the main part of the code that we want to test. Since the tests are running without a Sitecore context, the code cannot use the Sitecore API directly. Instead, it will use the interfaces provided by the Abstractions library you have just added to the solution.

When the code runs in the unit tests it will use the test implementations of those interfaces; when running as part of your Sitecore solutions it will use another implementation that wraps the Sitecore API. We therefore need to inject the correct implementation of the interfaces to this code.

Usually, you would leverage dependency injection to ensure your code is using the correct implementation of the interface. However, since this course does not cover how to configure dependency injection with Sitecore, you will adopt a simpler pattern known as Pure Dependency Injection. You will provide two constructors to the class, one with parameters which allow you to inject the implementation of the interface you want to use, this is the one you will use from your test code. You will add a second constructor, without parameters, that supplies a default implementation of the interface, this is the one that Sitecore itself will invoke.

Detailed Steps

1. Create a **new folder** named *Business* in the **events.tac.local project**.
2. Add a class named *BreadcrumbBuilder*.
3. Add *Pure DI* to inject a reference for the IRenderingContext interface. Don't forget to include the **using TAC.Sitecore.Abstractions.SitecoreImplementation** statement.

   ```
   public BreadcrumbBuilder() : this(SitecoreRenderingContext.Create()) { }
   ```

4. Create another constructor, accepting a parameter of type *IRenderingContext*, which initializes a readonly field of the same type.

5. Create a new method with the signature:

```
public IEnumerable<NavigationItem> Build()
```

```
public class BreadcrumbBuilder
{
    private readonly IRenderingContext _context;

    public BreadcrumbBuilder() : this(SitecoreRenderingContext.Create()) { }

    public BreadcrumbBuilder(IRenderingContext context)
    {
        _context = context;
    }

    public IEnumerable<NavigationItem> Build()
    {
        throw new System.NotImplementedException();
    }
}
```

> **NOTE:** You can set up Sitecore to use proper dependency injection, but it is not covered in this course.

6. Open the **Test Explorer** (Test > Windows > Test Explorer).
7. Click **Run All**. You should have 7 tests failing.

4. Writing the code for the for the Build method

Check the different tests defined in the **BreadcrumbBuilderTests class** to analyze the requirements. Write the code to make all tests pass. You can use the **IRenderingContext** to retrieve references to the context item and the home item. Notice those are **IItem interfaces**. You are avoiding any dependency on the Sitecore API. Check how the SitecoreItem class (in the TAC.Sitecore.Abstractions) is using the Sitecore API, including the LinkManager.

You should end up with a code similar to this:

```
public IEnumerable<NavigationItem> Build()
{
    return _context?.HomeItem == null || _context?.ContextItem == null ?

        Enumerable.Empty<NavigationItem>() :

        _context
            .ContextItem
            .GetAncestors()
            .Where(i => _context.HomeItem.IsAncestorOrSelf(i))
            .Select(i => new NavigationItem
                (
                    title   : i.DisplayName,
                    url     : i.Url,
                    active  : false
                ))
            .Concat(
                new[] {
                    new NavigationItem
                    (
                        title   : _context.ContextItem.DisplayName,
                        url     : _context.ContextItem.Url,
                        active  : true
                    )
                }
            );
}
```

5. Creating the Controller and View

Finally you need to create the controller and the view. The controller simply uses the **BreadcrumbBuilder** to create the model that is passed on to the view which then transforms it to HTML.

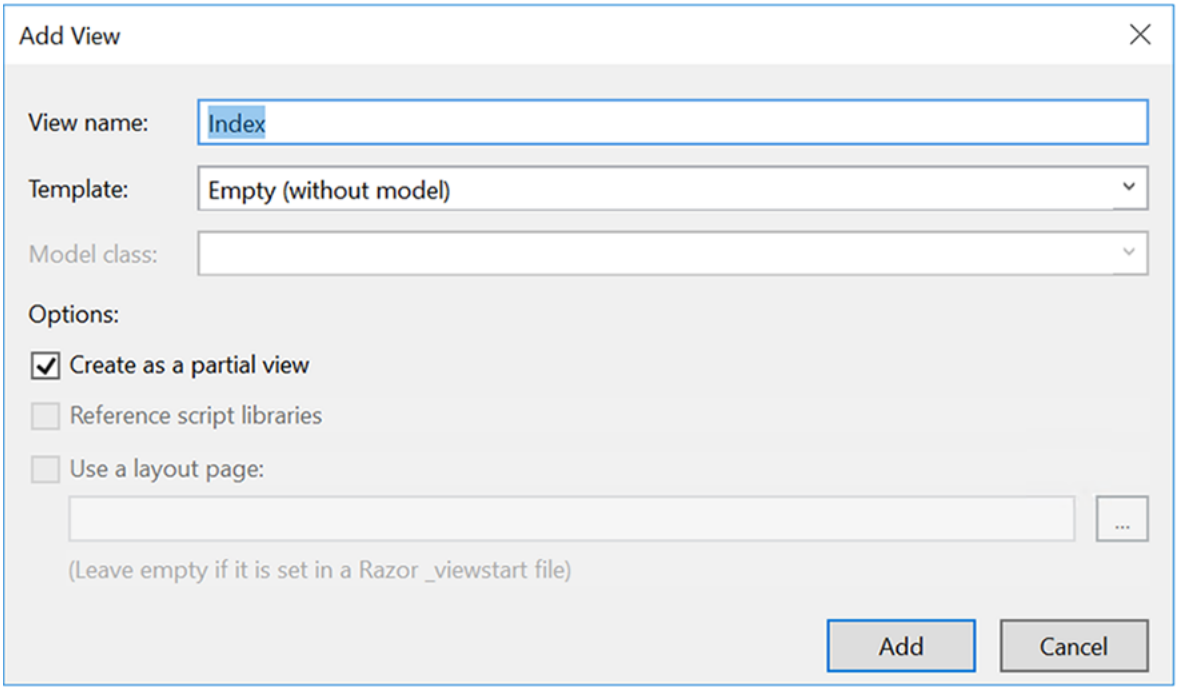
Detailed Steps

1. Create a new **Empty Controller** named *BreadcrumbController* in the **Controllers** folder.
2. Introduce a dependency on **BreadcrumbBuilder** (see code below).

3. Use the **BreadcrumbBuilder** to build the model and pass it on to the view:

```
public class BreadcrumbController : Controller
{
    private readonly BreadcrumbBuilder _builder;

    public BreadcrumbController() : this(new BreadcrumbBuilder()) { }

    public BreadcrumbController(BreadcrumbBuilder builder)
    {
        _builder = builder;
    }

    public ActionResult Index()
    {
        return View(_builder.Build());
    }
}
```

4. Right-click in the **Index method** and select **Add View...** to create a view.



5. Set the model for the view as *IEnumerable<NavigationItem>* using the @model directive at the top of the View code.
6. Copy the **HTML** displaying the Breadcrumb from the **Student Resource** folder. In the Event-Detail.html file, you can see the HTML that represents the Breadcrumb marked with the comment *[Breadcrumb Navigation]*.
7. Paste the **HTML** in the **View**.

8. Replace the hard-coded HTML with a **foreach loop** and references to the properties from the model.

```
@model IEnumerable<events.tac.local.Models.NavigationItem>
<!-- [Breadcrumb Navigation] -->
<ol class="breadcrumb">
    @foreach (var navItem in Model)
    {
        <li>
            <a class="@(navItem.Active ? "active" : string.Empty)"
               href="@navItem.URL">
                @navItem.Title
            </a>
        </li>
    }
</ol>
<!-- [/Breadcrumb Navigation] -->
```

> **NOTE:** You can add the events.tac.local.Models namespace to the web.config inside the Views folder to avoid using the full namespace of the class.

9. Save and deploy the code.

6. Creating the rendering definition Item and bind presentation:

Now that you have the code and all the files, you can create the definition item in Sitecore so that it can find the relevant resources in the files system and add them to the presentation details. Since you want this component to show up in all pages based on the Event Details template, you will add the component in the presentation details of the Event Details' Standard Values item.

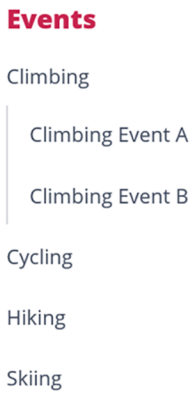
Detailed Steps

1. In the **Sitecore Explorer**, select */sitecore/layout/Renderings/TAC/Events* and create a new item based on the **Controller rendering** template named *Breadcrumb*.
2. Fill the **Controller** and ***Controller Action*** fields as *Breadcrumb* and *Index* respectively.
3. In the Sitecore Explorer, select the **Standard Values** for the **Event Details** template.
4. Open its presentation details by pressing **Ctrl+Enter.**
5. Click **Add Rendering** and select *Breadcrumb* component.
6. Press **Enter** to open its properties and set the **PlaceholderKey** as pageHeader.
7. Save your changes.
8. Preview an Event Details item in the Experience Editor to view the new breadcrumb component.

STOP HERE

Lab: Creating More Complex Navigation Structures

In this lab, you will build a more complex navigation structure: a menu. To view a sample, look at the Event-Detail.html page. It is a three level structure that shows links to pages represented by the Events Section, Events List and Event Details templates.



The easiest way to build this navigation structure, where only the direct ancestors are expanded, is by using a recursive method.

1. Creating the model

Start by creating a new model called *NavigationMenuItem* that stores the references to the children, that are based on **NavigationItem class**. This way you will be able to model the tree that forms the navigation menu.

Detailed Steps

1. In the **Models** folder, create a new class named *NavigationMenuItem.*
2. Make the class inherit from **NavigationItem**.
3. Add the following property with a **private set** and a **public get**.

Property Type	Property Name
IEnumerable<NavigationMenuItem>	Children

4. Add a constructor receiving parameters to populate the Title, URL and Children properties.

```
public class NavigationMenuItem : NavigationItem
{
    public NavigationMenuItem(string title, string url, IEnumerable<NavigationMenuItem> children) : base(title, url, false)
    {
        Children = children != null ? children : new List<NavigationMenuItem>();
    }

    public IEnumerable<NavigationMenuItem> Children { get; private set; }
}
```

2. Stub the model builder:

Now you will create a class responsible for creating the model that captures all the information required to render the menu. In this case, you need the title and URL for each entry in the menu. Generally, to make the component more generic, you would inject the item you want to use as the root of the menu either as a data source to the component, or added as a parameter. Alternatively, you can use properties like **TemplateID** to identify the root menu item while traversing the tree. In this lab, you will use the data source to specify the top level of the menu.

Detailed Steps

1. Create a new class named *NavigationBuilder* under the **Business** folder of the solution.
2. Use DI to add a dependency on IRenderingContext, just like you did with the BreadcrumbBuilder.
3. Stub a public Build method returning a **NavigationMenuItem object**.

```
public NavigationMenuItem Build(

{
        throw new NotImplementedException();
}
```

3. Importing the tests

You have some tests in the **NavigationBuilderTests.cs** file in the Student Resources folder. Import this class adding it to the tac.events.local.Tests project. Run the tests and you should have five failed tests and one skipped test from the NavigationBuilderDataTests.

4. Writing the code for the NavigationBuilder

You now must look at the tests and write the code for the Build method of the NavigationBuilder class so they don't fail. The best approach is to follow a **recursive strategy**; start with the root item (the data source item from the context), render all the children and follow the path of the only one that is an ancestor of the current item (the context item from the context).

Try building this first before looking at the solution!

```
public NavigationMenuItem Build()
{
    var root = _context?.DatasourceOrContextItem;

    return new NavigationMenuItem
    (
        title    : root?.DisplayName,
        url      : root?.Url,
        children : root != null && _context?.ContextItem != null ? Build(root, _context.ContextItem) : null
    );
}

private IEnumerable<NavigationMenuItem> Build(IItem node, IItem current)
{
    return node
        .GetChildren()
        .Select(i => new NavigationMenuItem
        (
            title    : i.DisplayName,
            url      : i.Url,
            children : i.IsAncestorOrSelf(current) ? Build(i, current) : null
        ));
}
```

Detailed Steps

1. Create a new Controller named *NavigationController* under the **Controllers** folder of the solution.
2. Using simple DI, add a dependency on **NavigationBuilder**. Use similiar code to that used for the Bread-crumbController.
3. Use the **NavigationBuilder** to produce the model. Pass it to the view.
4. Right-click the **Index** method and select ***Add View...*** to create a new empty view.
5. At the top of the View, add: **@model events.tac.local.Models.NavigationMenuItem** (you may prefer to omit the fully qualified reference to the model and add the **@using events.tac.local.Models;** statement instead).

5. Code the view

For the view, use recursion to render the menu. You can use the @helper directive, which allows you to define a method that renders some HTML and can be invoked multiple times.

Detailed Steps

1. From the **SideContent.cshtml file** cut the section of HTML between the comments <!-- [Page Navigation] --> and paste it onto the view.
2. Replace the code in the first **H4 header** so that the link and the title come from the model.
3. Below all the code create a **new helper method**:

```
@helper RenderMenu(NavigationMenuItem navMenu)
.
```

4. Move the ** tag** and all its content inside the method.
5. Delete all the ** tags** (including the inner <ul>), except for the last one.
6. Create a **foreach loop** to iterate over all the child items of the parameter of the method and put the remaining ** tag** inside the loop.
7. Replace the link and text in the <a> tag with the **URL and Title properties** of the object iterated in the foreach loop.
8. Add a class attribute and add a **class open** if the object's Children property is not null or *close* otherwise.
9. After the <a> tag, but still inside the ** add @RenderMenu** (to invoke the helper method), but pass the object being enumerated.
10. Inside the ** tag** above the helper method, add **@RenderMenu** but passing the model.
11. Compile and deploy all your changes.

Your code may look like this:

```
@model NavigationMenuItem

<!-- [Page Navigation] -->
<h4><a href="@Model.URL">@Model.Title</a></h4>
<div class="sidebar sidebar-static">
    @RenderMenu(Model)
</div>
<!-- [/Page Navigation] -->
@helper RenderMenu(NavigationMenuItem navMenu)
{
    if (navMenu.Children != null)
    {
            <ul class="nav nav-pills nav-stacked">
                @foreach (var menuEntry in navMenu.Children)
                {
                    <li class="@(menuEntry.Children != null? "open" : "") ">
                        <a href="@menuEntry.URL">@menuEntry.Title</a>
                        @RenderMenu(menuEntry)
                    </li>
                }
            </ul>
    }
}
```

6. Bind the presentation

Create a definition item for the newly created controller and add it to the presentation details of the Standard Values for the Event Details template. This way the new component will show up on all Event Details pages.

Detailed Steps

1. Create a new item under */sitecore/layout/Renderings/TAC/Events*. Base the item on the **Controller**

rendering template and name it *NavigationMenu.*

2. Fill the **Controller** and **Controller Action** fields so they point to your controller and save.
3. Select the */sitecore/templates/TAC/Events/Event Details/__Standard Values* item and press **Ctrl+Enter** to edit the presentation details.
4. Click **Add Rendering** on the toolbar.
5. Search for the **Navigation Menu** component and click **OK.**
6. Select the **Navigation Menu** that you just added and press **F4** to open the Properties window.
7. Set the PlaceholderKey to *pageSide.*
8. Set the Data source (also in th eProperties window) to the */sitecore/content/home/events* item.

> **NOTE:** Use the item picker button to the right of the textbox, or paste the ID. Avoid using paths as data sources.

9. **Save** all your changes.
10. Using the **Preview** application, navigate to an **Event Details** item and verify that the navigation menu displays the correct items.

STOP HERE

MODULE 6: Configuring and Extending Sitecore

In this module, you will learn how to customize Sitecore by understanding the Layers of configuration and by applying specific patch files to the Sitecore section of the web.config file located in the App_Config/Include folder. You will learn how to extend Sitecore with packages, and how to configure multiple sites in one Sitecore instance.

Scaling is also covered, with specific attention given to Sitecore server functions through **roles**.

Lab: Installing a Package

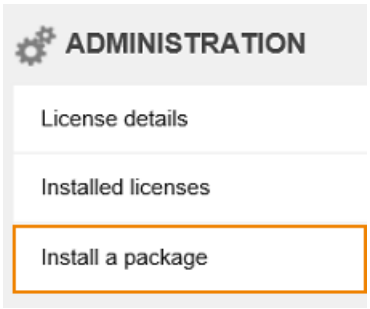
In this lab, you will add other files and items to the solution by installing a package.

You can install packages through Sitecore Rocks or through the Sitecore interfaces. Sitecore Rocks does not prompt the action to take in case of conflict, it simply overrides. The Sitecore interface asks the user by default.

You are going to install a package that contains the items and files necessary to run another site for the Adventure Company. Because the package will have conflicts, you will use the Sitecore UI.

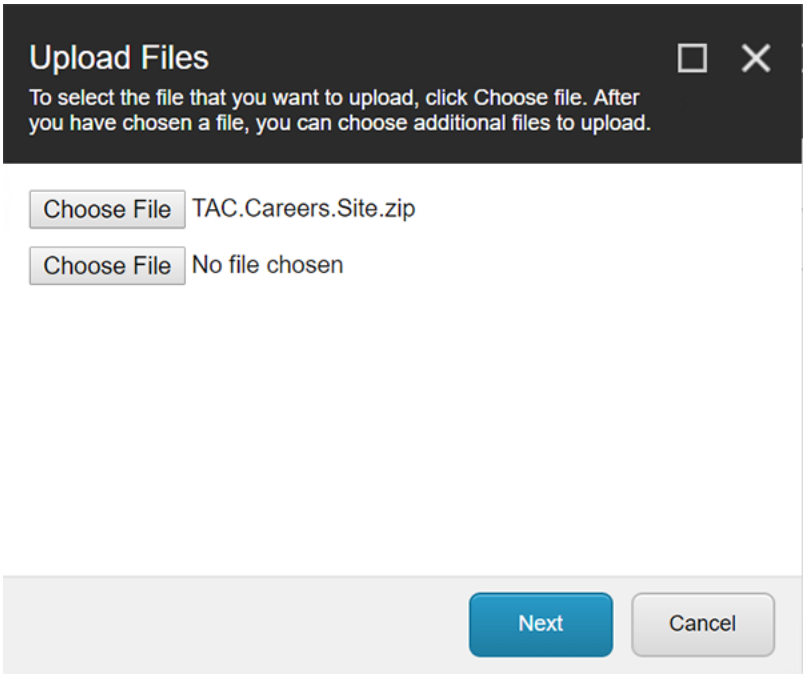
Detailed Steps

1. In your browser, navigate to http://events.tac.local/sitecore and log in to the Sitecore interface.
2. On the **Launchpad**, open the **Control Panel**.
3. In the Administration section, click the **Install a package** option.

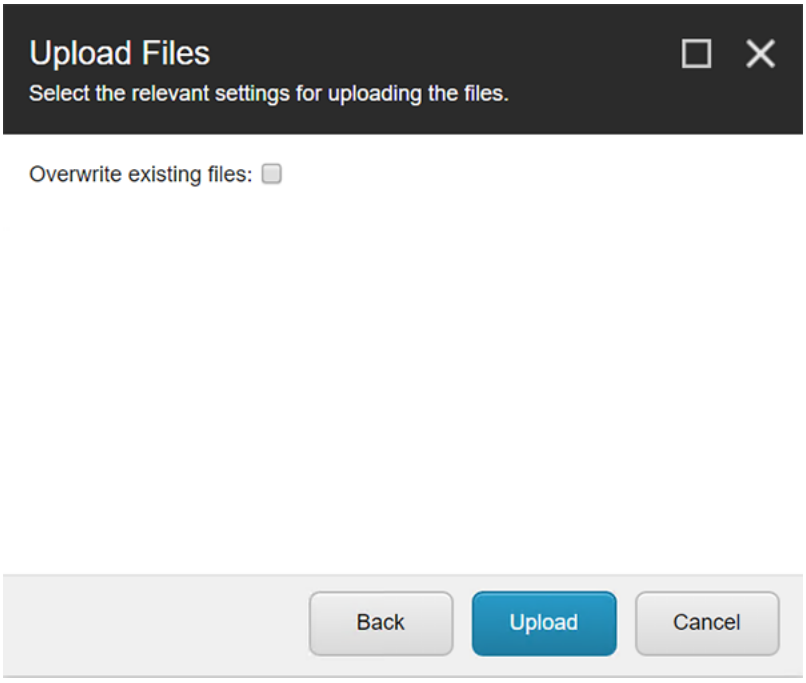


4. Click **Upload package**
5. Depending on which browser you are using, click **Choose File** or **Browse**.

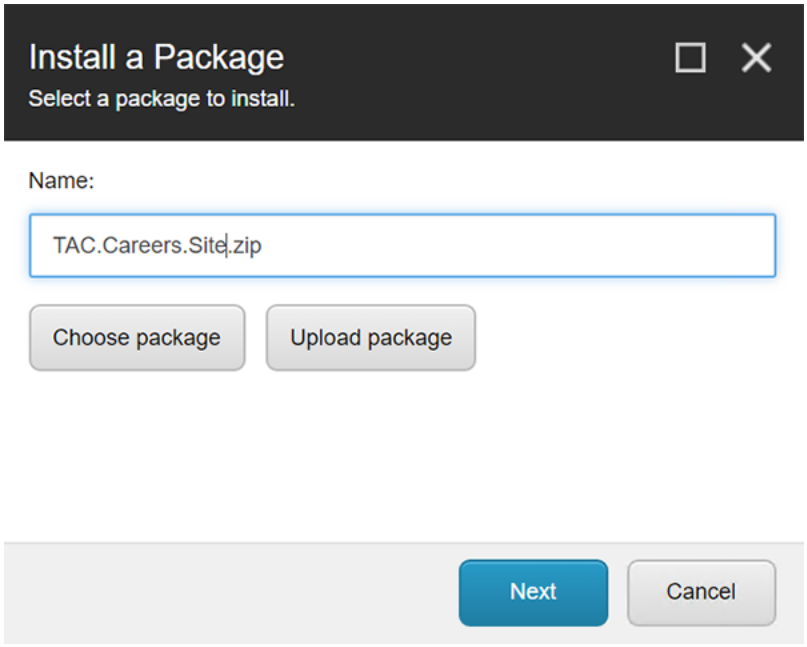
6. In the **Student Resources** folder, select the **TAC.Careers.Site.zip** and then click **Next**.



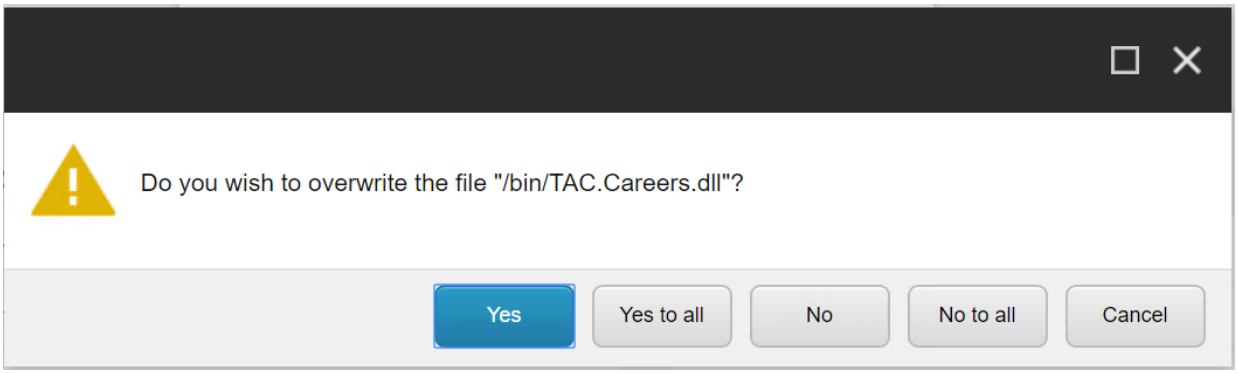
7. Click **Upload**.



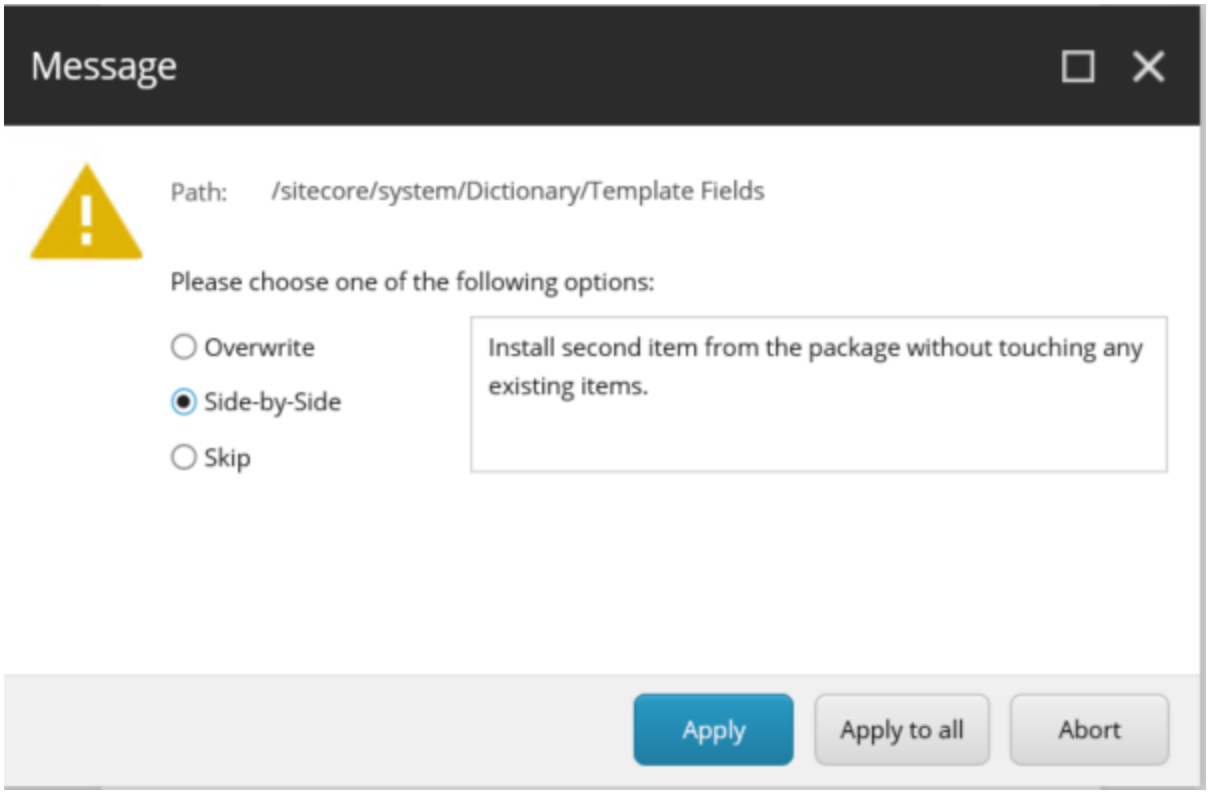
8. Click **Close** and then click **Next**.



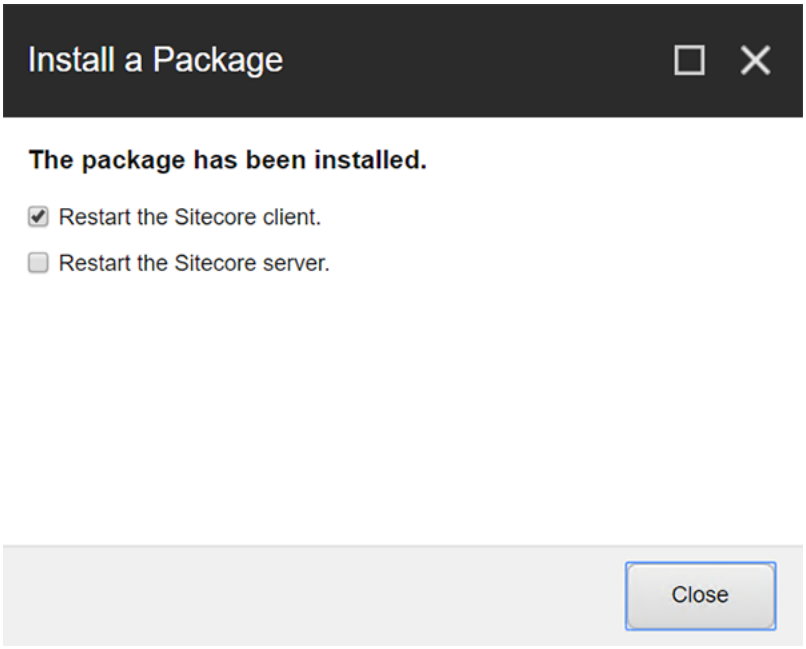
9. You will see the details about the package. Click **Install** to proceed.
10. If you get a prompt for a **file** conflict, click **Yes to all**.



11. In the first prompt of **item** conflict, choose **Overwrite**. Click **Apply to all**.



12. On the screen confirming the installation has finished, click **Close**.

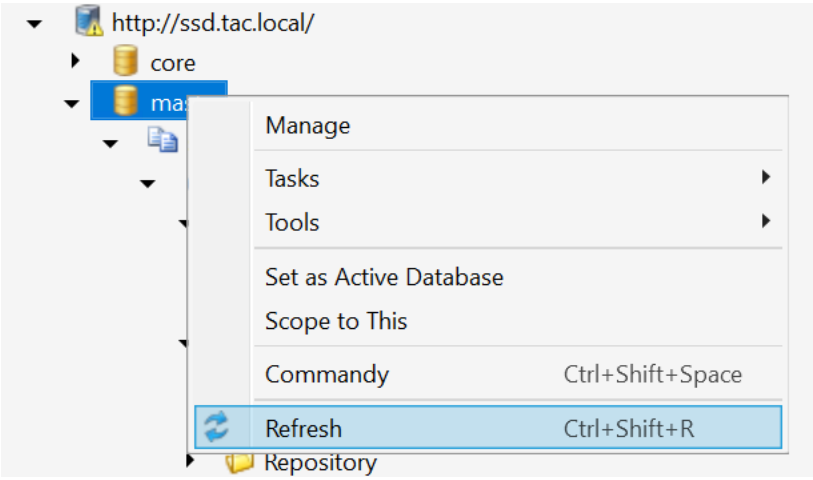


1. Revising the installed items

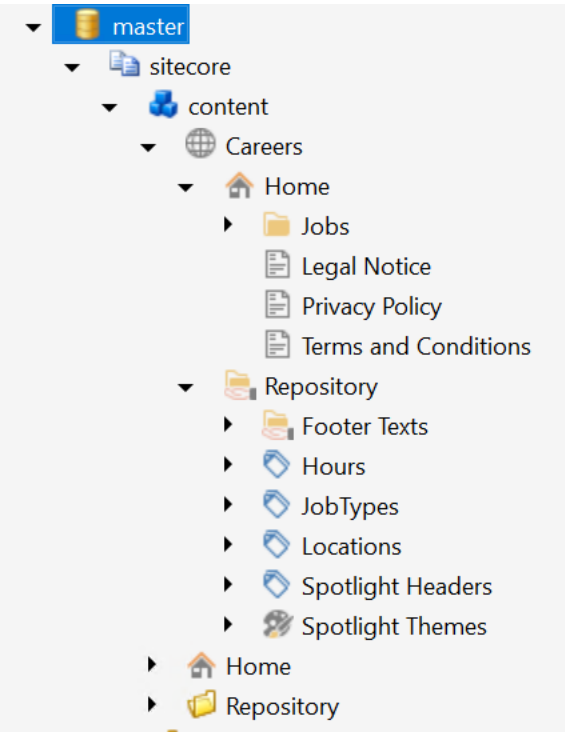
If your package has installed correctly, you will see some new items in the content tree.

Detailed Steps

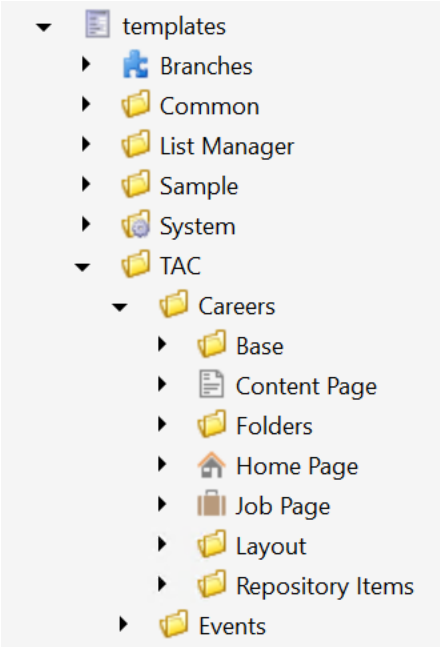
1. Open the **Sitecore Explorer** in Visual Studio.
2. Right-click the **Master database node** and choose **Refresh**.



3. Expand the content tree and explore the new items under content.



4. Explore the new templates.



2. Inspecting the new configuration files

Detailed Steps

1. In the **File Explorer**, open the webroot for the Sitecore installation.
2. Navigate to the *App_Config\Include\TAC* folder. Notice and inspect the new files that it contains.

> **NOTE:** Alternatively you could unscope Sitecore Rocks from Master and view the *Website\App_Config\Include\TAC folder* in Sitecore Explorer.

STOP HERE

Lab: Configuring Sitecore for Multiple Sites

In this lab, you will change Sitecore's configuration so it renders content from different parts of the content tree according to the hostname of the request. This is how Sitecore supports multiple sites in a single installation.

The package you installed in the last lab has added a configuration Include file to enable Sitecore to show the new Careers site. However, this is not enough. You also need to enable this hostname in your hosts file and add it to the IIS bindings. You can do this using PowerShell.

Detailed Steps

1. Open a **PowerShell** window as administrator.

   > **NOTE:** You can press the Microsoft key, search for PowerShell and press **Ctrl+Shift+Enter**on the keyboard.

2. To add the binding to the IIS site execute:

   ```
   New-WebBinding -name tac.corporate -HostHeader careers.tac.local
   ```

   > **NOTE:** You will get an error if that binding is already defined.

3. To add the host name to the file, you can use Notepad. Run the following command in **PowerShell**:

   ```
   notepad "$env:windir\system32\drivers\etc\hosts"
   ```

3. Add a new line to the file.

   ```
   127.0.0.1       careers.tac.local
   ```

4. Save and exit **Notepad**.
5. Close the **PowerShell** window.

1. Browse the Careers site

Before you can browse the site, you will need to perform a Publish operation.

Detailed Steps

1. In **Visual Studio**, using the **Sitecore Explorer**, select any item in the Master database.
2. Open **Commandy** with the shortcut **Ctrl+Shift+Space**.
3. Type *smart* and press **enter**. This will activatre a smart Publish operation for the whole database.
4. Open a new browser window and open careers.tac.local

2. Move the Events site to a folder

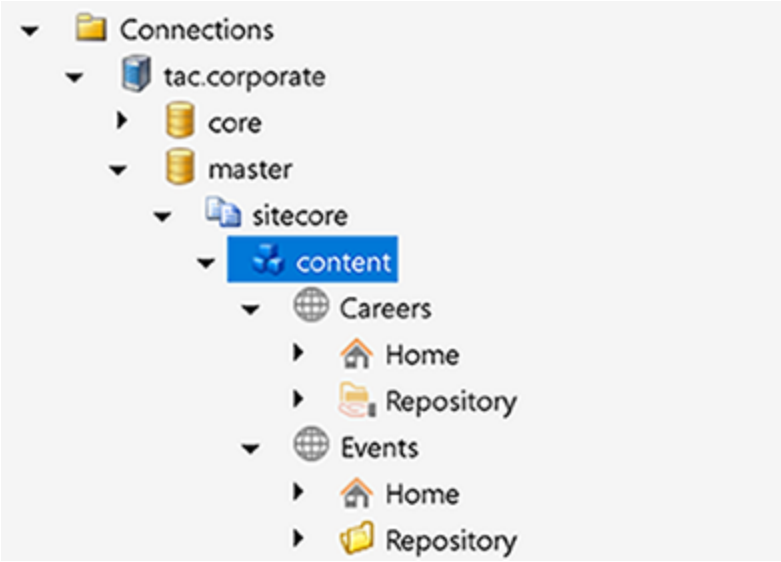
You will now change your Events site so it resides in its own folder, just like the Careers site does.

Detailed Steps

1. Create a new **template folder** item named *Common* under the */sitecore/templates/TAC* folder.
2. Move the */sitecore/templates/TAC/Careers/Folders/Site* folder item into the **Common Template** folder; you can simply drag and drop it in the Sitecore Explorer.

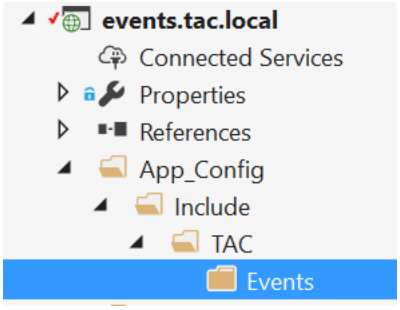
   > You are going to reuse the Site Folder template for both sites (Events and Careers), therefore, you are placing it in a folder outside of those of the sites. You may notice that there are other templates that could have been shared, like the base templates. You can now change the template inheritance so that the templates from both sites inherit from a common set of templates. But if you do this, you will lose the data you have already added. Of course, if this had been done before creating content, it would not be an issue.

3. Create a new item named *Events* under /sitecore/content based on the **Site Folder** template.
4. Move the */sitecore/content/Home* and */sitecore/content/Repository* items inside the **Events** folder. Your content tree should like this:

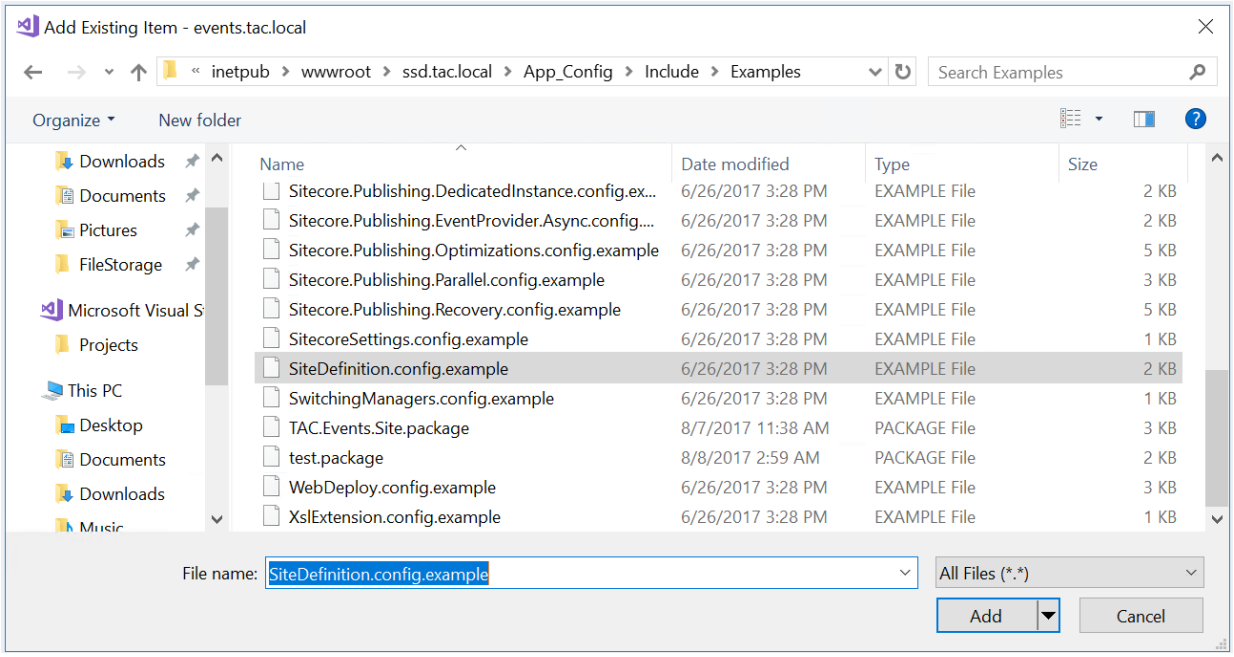


5. Perform a **Smart Publish** operation.

6. In the Solution Explorer, create a new folder structure: *App_Config\Include\TAC\Events*.



7. Right-click the new **Events** folder and choose **Add, Existing Item...**
8. Select the **SiteDefinitionItem.config.example** from the *App_Config\Include\Examples* folder inside the webroot



.

9. Rename the item *events.site.config*.
10. Change the **rootPath** attribute to */sitecore/content/Events*.
11. Change the **name** attribute to *events*.
12. Add a new attribute **hostName** with the value **events.tac.local**.

13. Add an attribute **scheme** with value **http**.

> **NOTE:** When Sitecore builds full URLs with the hostname, it uses this scheme attribute.

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <sites>
      <site name="events" patch:before="site[@name='website']"
            virtualFolder="/"
            physicalFolder="/"
            rootPath="/sitecore/content/events"
            startItem="/home"
            hostName ="events.tac.local"
            scheme="http"
            database="web"
            domain="extranet"
            allowDebug="true"
            cacheHtml="true"
            htmlCacheSize="50MB"
            enablePreview="true"
            enableWebEdit="true"
            enableDebugger="true"
            disableClientData="false"/>
    </sites>
  </sitecore>
</configuration>
```

14. Save and deploy this file.

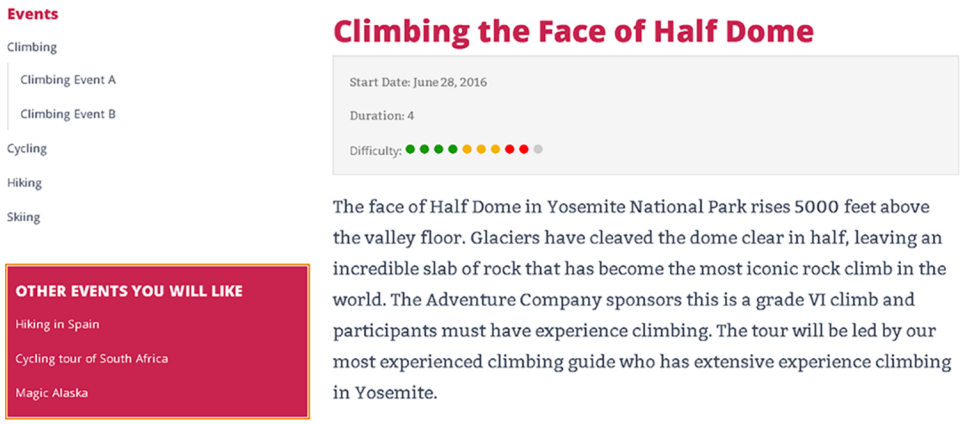
STOP HERE

MODULE 7: Working with Complex Field Types

In this module, you will create a new component that can be used to display related events from a complex field type. You will also learn how to render complex field types in the Experience Editor.

Lab: Rendering Complex Field Values

In this lab, you will add a new field to the Event Details template and create a component to display a list of related events. This allows authors to promote a list of events on the left-hand side of an Event Details page.



To add this new functionality, you need to handle both the **data** and the **presentation**. For data, you will add a new field in the Event Details template to store the related events. For presentation, you will create a new component to display the data and dynamically add it to the page in the presentation details for Event Details items.

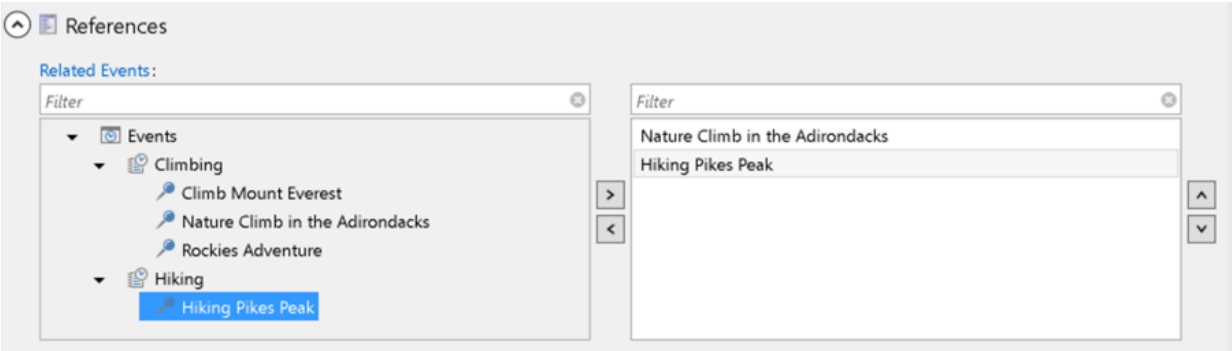
1. Modifying the Event Details data template:

Detailed Steps

1. Using **Sitecore Rocks**, locate the **Event Details** data template.*(/sitecore/templates/TAC/Events/Event Details*) and double-click to edit it.
2. Create a new field section called *References*.
3. In the new field section, add a field called *RelatedEvents* of the type *Treelist.*
4. Set the source of the new field to point to */sitecore/content/Events/Home/Events*.

5. Press **Ctrl+S** to save your changes.
6. Navigate to one of the **Event Details** items and verify that the new field appears in the References field section.

7. Select a few related events and click **Save**.



2. Add the tests

Detailed Steps

1. In the **Business** folder, create a new class named *RelatedEventsProvider*.
2. Add simple dependency injection to add a dependency on a *IRenderingContext* implemented by *SitecoreRenderingContext*, as you did for the BreadcrumbBuilder and NavigationBuilder classes in earlier labs.
3. Add a new method with the signature:

```
public IEnumerable<NavigationItem> GetRelatedEvents()
```

4. Leave the method block with a:

```
 throw new NotImplementedException();
```

.

5. In the events.tac.local.Tests project use the **Add, Existing item** to include the **RelatedEventsProviderTests.cs** file from the Student Resources folder.
6. In the **Text Explorer** window, run all tests. You should have three RelatedEvents tests failing since you have not yet implemented the **GetRelatedItems** method.

3. Implement the code to retrieve related events

Inspect the code in the RelatedEventsProviderTests to understand the requirements and then implement the code of the GetRelatedEvents method.

Detailed Steps

1. In the **GetRelatedEvents** method of the **RelatedEventsProvider** class:
    - Retrieve the **context item** from the IRenderingContext.
    - Use the **GetMultilistFieldItems** method to retrieve the items referenced in the **RelatedEvents** field.

- Project the items returned into objects of type **NavigationItem**. Assign *DisplayName property* as the Title.

2. Refine your code until all tests are passing.

Your code may look similar to:

```
public IEnumerable<NavigationItem> GetRelatedEvents()
{
    return _context
        .ContextItem
        .GetMultilistFieldItems("relatedEvents")
        .Select(i => new NavigationItem
        (
           title : i.DisplayName,
           url   : i.Url
        ));
}
```

Inspect the GetMultilistFieldItems method of the SitecoreItem class, to see how to use the Sitecore API to manipulate a MultilistField.

4. Creating the new Related Events Controller:

Detailed Steps

1. In the **Solution Explorer** of Visual Studio, right-click the **Controllers** folder and select **Add > Controller...**
2. Select **MVC 5 Controller - Empty** and click **Add.**
3. Name the class *RelatedEventsController*. Visual Studio will scaffold a new class for the controller.
4. Add a dependency to **RelatedEventsProvider.**
5. Get the model generated by the method in the **RelatedEventsProvider** class.

6. Change the return statement so the model produced by RelatedEventsProvider is passed on to the **View**. Your code should look similar to this:

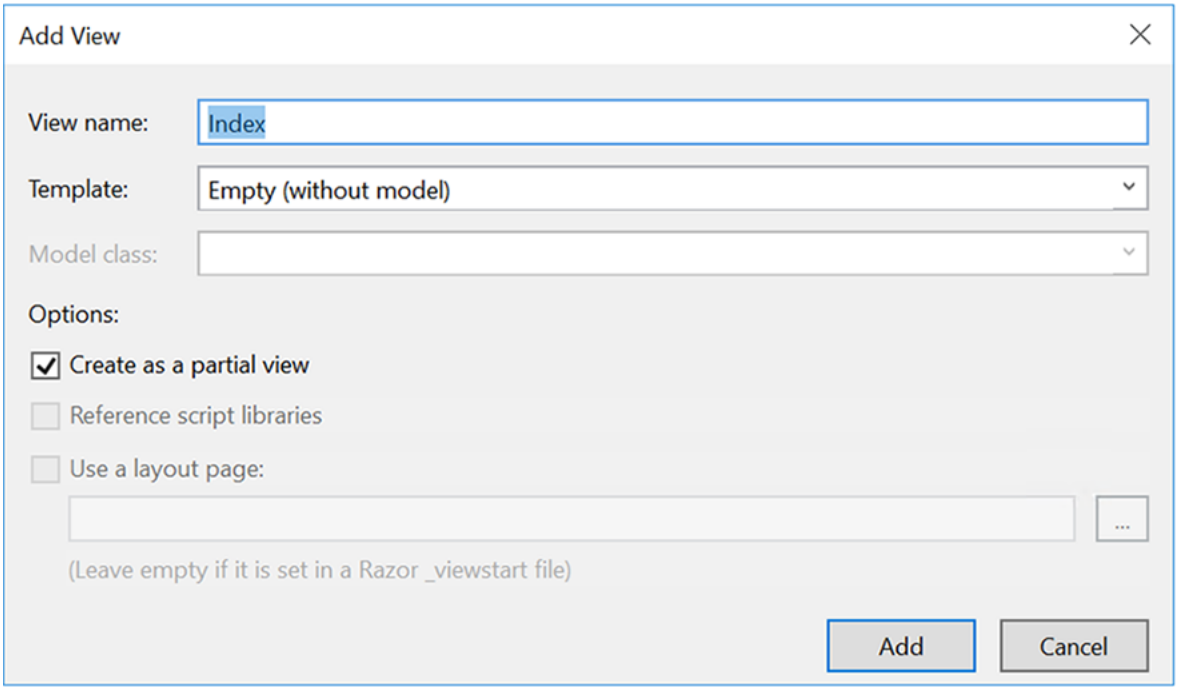
```
public class RelatedEventsController : Controller
{
    private readonly RelatedEventsProvider _provider;

    public RelatedEventsController() : this (new RelatedEventsProvider()) { }

    public RelatedEventsController(RelatedEventsProvider provider)
    {
        _provider = provider;
    }

    public ActionResult Index()
    {
        return View(_provider.GetRelatedEvents());
    }
}
```

5. Creating the View:

Detailed Steps

1. Right-click the **Index** method and select **Add View...**
2. Make sure that **Create as Partial View** is enabled.

3. In the **Template** drop-down menu, select **Empty (without model)**:



4. Add a **@model declaration** at the top of the view for *IEnumerable<events.tac.local.Models.NavigationItem>* .
5. Open the **SideContent.cshtml View Layout** and locate the part of the HTML responsible for rendering the [Related Events] part of the page.
6. Cut the code and paste it on to the **View.**

```
@model IEnumerable<events.tac.local.Models.NavigationItem>
<!-- [Related Events] -->
<div class="bg-primary side-widget">
    <h4 class="text-uppercase">Other events you will like</h4>
    <div class="sidebar">
        <ul class="nav nav-pills nav-stacked">
            <li>
                <a href="#">Hiking in Spain</a>
            </li>
            <li>
                <a href="#">Cycling tour of South Africa</a>
            </li>
            <li>
                <a href="#">Magic Alaska</a>
            </li>
        </ul>
    </div>
</div>
<!-- [/Related Events] -->
```

7. Add a **foreach loop** iterating over the model to render the li tags.

8. Replace the code in the anchor tags so it gets the information from the model.

```
@model IEnumerable<events.tac.local.Models.NavigationItem>
<!-- [Related Events] -->
<div class="bg-primary side-widget">
    <h4 class="text-uppercase">Other events you will like</h4>
    <div class="sidebar">
        <ul class="nav nav-pills nav-stacked">
            @foreach (var navItem in Model)
            {
            <li>
                <a href="@navItem.URL">@navItem.Title</a>
            </li>
            }
        </ul>
    </div>
</div>
<!-- [/Related Events] -->
```

9. Save and deploy all the changes to the Sitecore instance.

Before you can add the component on to the page, you need to create the definition item of the component, so you can select it on the presentation details.

6. Creating a component definition item:

Detailed Steps

1. In **Sitecore Explorer**, navigate to */sitecore/layout/Renderings/TAC/Events.*
2. Invoke **Commandy** by pressing **Ctrl+Shift+Space**.
3. Press ***i*** followed by a space. Commandy will enter **insert mode** and show the available insert options.
4. Select **Controller Rendering** and name it *RelatedEvents***.**
5. Change the **Controller field** to *RelatedEvents* and change the **Action** to *Index*.
6. Save the changes pressing **Ctrl+S**.

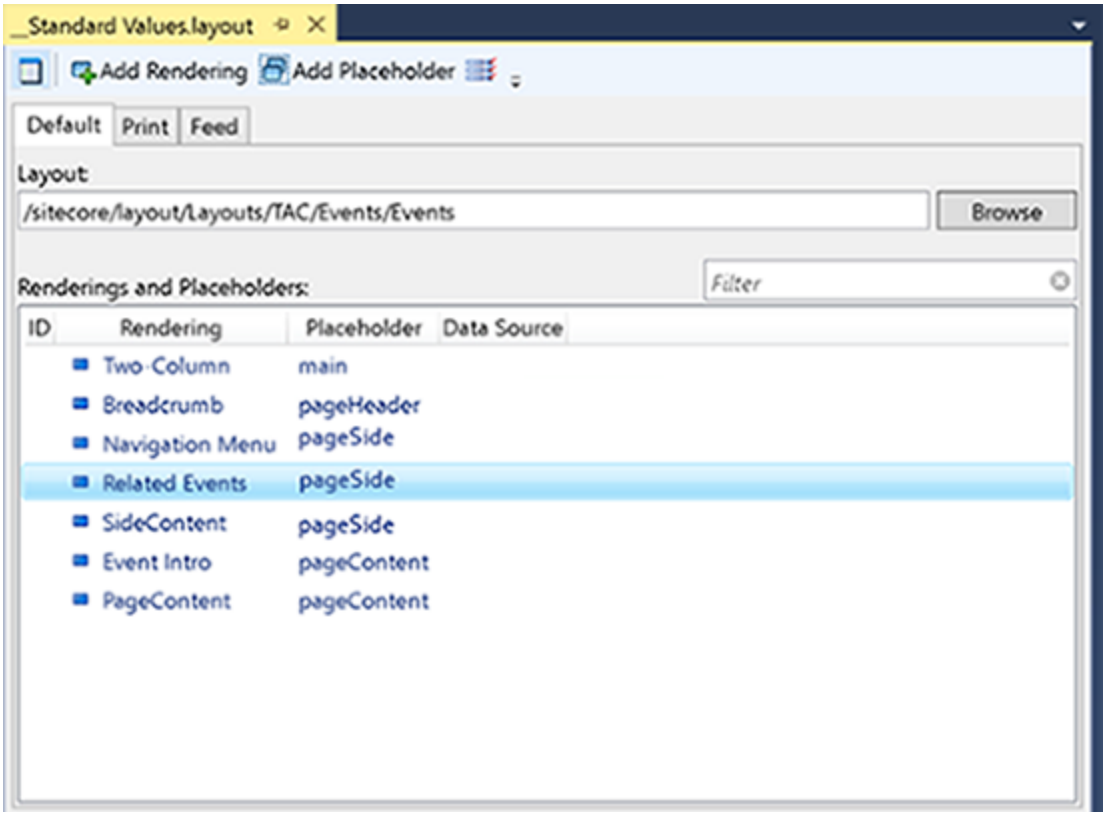
To show this component on the page, bind it dynamically by placing it in a placeholder and by adding it to the presentation details. Since you want the component on every Event Details page, add it to the **Standard Values**.

7. To bind the component to the page:

Detailed Steps

1. In Sitecore Explorer, locate and select the **Standard Values** of the Event Details template (*/sitecore/templates/TAC/Events/Events Details/__Standard Values*).
2. Open the **Presentation Details** by pressing **Ctrl+Enter.**
3. Select **Add Rendering.**

4. Choose the **Related Events** component that you just created.
5. Select the component and press **F4** to show its properties.
6. In the **Placeholder key field,** enter *pageSide*



7. Save your changes.
8. Preview an **Event Details** page and ensure the component appears on the page. See the first image of this lab to compare the rendering.

STOP HERE

Lab: Using Edit Frames to Edit Complex Fields in the Experience Editor

In this lab, you are going to add an edit frame around the Related Events component that you created earlier. On that edit frame, you will add a Field Editor button that will allow authors to modify the Related Events field directly from the Experience Editor.

Detailed Steps

1. On the Sitecore Explorer in Visual Studio, expand the **Core** database and locate the item: */site-core/content/Applications/WebEdit/Edit Frame Buttons*.

   > **NOTE:** If you've scoped the Sitecore content tree to Master database, right-click and choose **Unscope This** to show the Core database.

2. Create an item based on the **Edit Frame Button Folder** template and call it *Related Events*.
3. Under the **Related Events** folder, create another item based on the Field Editor Button template called *Edit Related Events.*
4. In the Edit Related Events item:
   - Choose an Icon by clicking **Browse**. For example: *Office/16x16/link.png*
   - Fill in the **Fields** field with the text *RelatedEvents*. This is the name of the complex field that authors will be able to modify inside the **Field Editor** window.
5. Save the changes.
6. Open the **Index** view corresponding to the **RelatedEvents** controller located in */Views/RelatedEvent-s/Index.cshtml.*
7. At the top of the code module, add a reference to **Sitecore.Mvc.Extensions** with a *using* statement.
8. Surround the whole HTML with a using a block as follows:

```
@using (Html.EditFrame(RenderingContext.Current.ContextItem.Paths.FullPath, "Related Events", "Edit Related Events", null, null, null))
{
    //Existing HTML
}
```

```
@using(Html.EditFrame(RenderingContext.Current.ContextItem.Paths.FullPath
,"Related Events","Edit Related Events",null,null,null))
{
        //Existing HTML
}
```

9. Deploy the **View** file by pressing **Alt+; Alt+P.**
10. Open the **Experience Editor**. Ensure that you are in **Editing** mode and open one of the **Event Details** pages. Check that you can now edit the **Related Events** field.

    > **NOTE:** When you use the Experience Editor to modify the value of a complex field, you need to save your changes. This refreshes the browser and your saved modifications will be rendered.

STOP HERE

MODULE 8: Implementing Search Driven Components

In this module, you will learn about search features of the UI. You will also learn how to use buckets to store large amounts of data and how to use that ContentSearch API directly in your own code.

Lab: Searching for Content

In this lab, you will explore the searching options available in the Sitecore UI.

1. To perform a simple search:

Detailed Steps

1. Open the **Content Editor**.
2. Select the */sitecore/content/Events/Home* item. This will force a search to only returns subitems of Home.
3. Click the **magnifying glass** icon, next to the Content tab, to open a **Search box**.
4. In the Search box, type *Climb* and run the search.
5. Click the **View** icons beside the Search box and notice the changes.

> **NOTE:** As a developer, you can create custom views that are defined under */sitecore/system/Settings/Buckets/Views*.

2. Complex queries

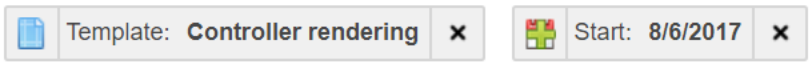
Try a few more complex queries now by executing each of the following:

Detailed Steps

1. Search for items under */sitecore/content/Events* that contain the term, **Climb** but not the term **Everest**.

2. Search for any item in the content tree based on the **Country** template that starts with the letters **Sp**.

3. Search all **controller renderings** created in the last 10 days. As per the previous search task, use the top level Sitecore node in the content tree for this search's starting point.



STOP HERE

Lab: Creating a Custom Index

For this lab, you will create a new index that contains the data of a certain part of the content tree. You may need to build a new index because:

- You want a certain part of the content tree to be **isolated** inside an index by itself.
- You need to bring in external data into Sitecore so that it is **searchable** within Sitecore.
- You need to **shard indexes** to support large amounts of content.

> **NOTE:** This lab requires that you have installed Solr as your default search provider. If you want to use a search index that works in both Analytics and ContentSearch, Sitecore recommends that you use Solr.

1. Create the common configuration

Because you are going to create an index for Events in the Master database and another one for the Events in the Web database, you will create a common configuration. This configuration is shared between the two new indexes, defining *what* items will be indexed and *how* they will be indexed.

Detailed Steps

1. Copy the **Events.DefaultIndexConfiguration.config** from the Student Resources folder, into the */App_Config/Include/TAC/Events* folder of your solution.

   > **NOTE:** In Sitecore versions prior to 9, configuration layers where not available. All Sitecore configuration files and folders were loaded alphabetically from the App_Config/Include folder. It was common to see custom files and folders prefixed with **Z**, for example, *App_Config/Include/ZZZ/zzz.config*, so that changes were not overridden.

2. In **Visual Studio**, you will need to **Show All Files**. Use the **Include in Project** feature and then open the file for editing.
3. Inside <include hint="list:AddIncludedTemplate"> element, add an element for the templates that you want to index, in this case just **Event Details.**

   <EventDetails>[YOUR TEMPLATE GUID HERE]</EventDetails>

4. Inside the <include hint="list:AddIncludedField"> element, add elements for each field that you want to index. In this case, add the following fields:**ContentHeading, ContentIntro, StartDate**

   <contentheading>[YOUR FIELD GUID HERE]</contentheading>.

   > **NOTE:** When you add template GUIDs and field GUIDs, the actual name of the element is irrelevant. Use something appropriate to identify *what* the GUID is. It is only the value inside the element that Sitecore uses.

5. Inside the <fieldNames hint="raw:AddFieldByFieldName"> element add **<field>** elements for each of the fields added above, indicating the returnType:
   - contentheading - text
   - contentintro - text
   - startdate- datetime

```
<field fieldName="contentheading" returnType="text" />
<field fieldName="contentintro" returnType="text" />
<field fieldName="startdate" returnType="datetime" format="yyyy-MM-dd'T'HH:mm:ss'Z'" />
```

   > **NOTE:** Inside <fieldMap> node, use the Sitecore field name with lowercase.

6. Save the changes.

2. Create the definition for the indexes

Detailed Steps

1. Copy the file *Sitecore.ContentSearch.Solr.Index.Web.config* from the *App_Config/Sitecore/ContentSearch* folder of your Sitecore installation.
2. Paste the file in the *App_Config/Include/TAC/Events* folder of your solution.
3. Rename the file *Solr.Events.Web.Index.config.*
4. In **Visual Studio**, you will need to **Show All Files**. Use the **Include in Project** feature and then open the file for editing.
5. In the element **<index>**, change the ID attribute to *events_web_index.*
6. Change the <param desc="core"> to point to ***tac.corporate_events_web***.
7. Locate the element **<configuration>** that is inside the **<index>** element.
8. Change its **ref attribute** so it is as follows:

   <configuration ref="contentSearch/indexConfigurations/defaultEventsIndexConfiguration"/>

9. Locate the element <Root> inside the <locations> element and change its value to */sitecore/content/Events/Home/Events.*

```
<contentSearch>
  <configuration type="Sitecore.ContentSearch.ContentSearchConfiguration, Sitecore.ContentSearch">
    <indexes hint="list:AddIndex">
      <index id="events_web_index" type="Sitecore.ContentSearch.SolrProvider.SolrSearchIndex, Sitecore.ContentSearch.SolrProvider">
        <param desc="name">$(id)</param>
        <param desc="core">TAC.SSD_events_web</param>
        <param desc="propertyStore" ref="contentSearch/indexConfigurations/databasePropertyStore" param1="$(id)" />
        <configuration ref="contentSearch/indexConfigurations/defaultEventsIndexConfiguration" />
        <strategies hint="list:AddStrategy">
          <strategy ref="contentSearch/indexConfigurations/indexUpdateStrategies/onPublishEndAsyncSingleInstance" />
        </strategies>
        <locations hint="list:AddCrawler">
          <crawler type="Sitecore.ContentSearch.SitecoreItemCrawler, Sitecore.ContentSearch">
            <Database>web</Database>
            <Root>/sitecore/content/Events/Home/Events</Root>
          </crawler>
        </locations>
        <enableItemLanguageFallback>false</enableItemLanguageFallback>
        <enableFieldLanguageFallback>false</enableFieldLanguageFallback>
      </index>
    </indexes>
  </configuration>
```

10. Save the file.
11. Duplicate the file and name it *Solr.Events.Master.Index.config.*
12. Use **Show All Files** and the **Include in Project** feature and open the file for editing.
13. Change the **<index> id attribute** so it is called *events_master_index.*
14. Change the **<database>** inside the <crawler> element so it points to **master.**
15. Change the **<param desc="core">** to the value *tac.corporate_events_master*.
16. Change the **<strategy>** element to: *ref="contentSearch/indexConfigurations/indexUpdateStrategies/**syncMaster**"*.
17. Save the changes and deploy all three files.

3. Adding the new cores to Solr

You need to create the new Solr cores. These instructions may vary slightly according to how you installed Solr.

Detailed Steps

1. Find the location for the **Solr cores** (generally /server/solr in your Solr installation folder).
2. Stop the Solr service using the **Administrative Tools/Services application**.
3. Copy the **tac.corporate_master_index core** folder and paste it into the same place.

> **NOTE:** The exact name may change depending on how you installed Sitecore.

4. Rename the new folder to *tac.corporate_events_master*.
5. Delete the contents of the **tac.corporate_events_master\data folder**.
6. In the root of the the tac.corporate_events_master folder, open the **core.properties file** using a text editor and change the name setting to match the folder name.

7. Save and close the file.
8. Duplicate the **tac.corporate_events_master** and name it *tac.corporate_events_web*.
9. In the root of the the tac.corporate_events_web folder, open the **core.properties file** using a text editor and change the name setting to match the folder name.
10. Save and close the file.
11. Start the Solr service using the **Administrative Tools/Services application**.

4. Publish the website

Before rebuilding the indexes you will perform a Publish operation. This will ensure the Web database contains the most current content.

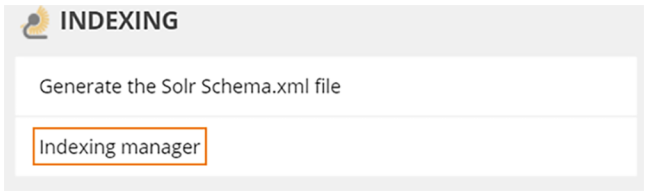
Detailed Steps

1. In **Visual Studio**, using the **Sitecore Explorer**, select any item in the Master database.
2. Open **Commandy** with the shortcut **Ctrl+Shift+Space**.
3. Type *smart* and press *enter*.

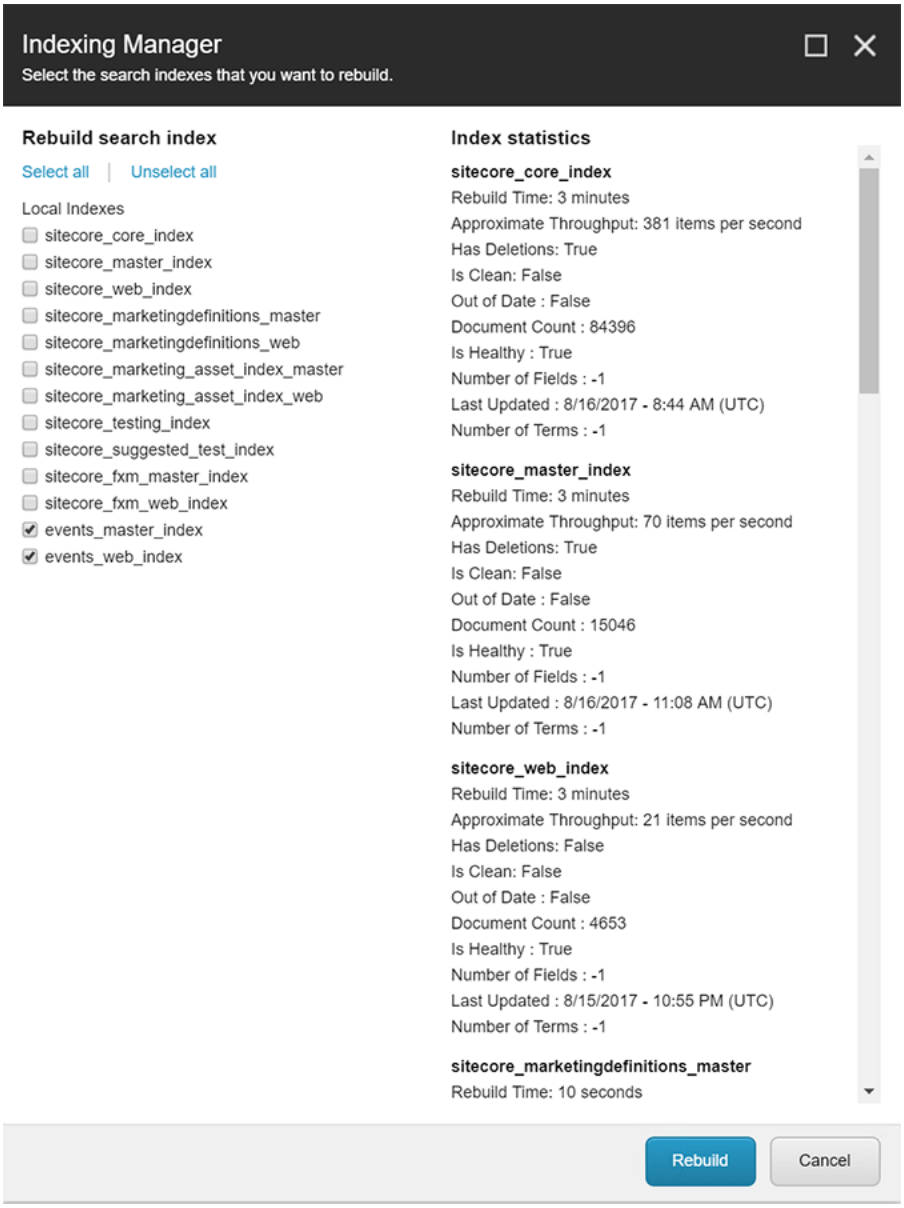
5. Rebuilding the indexes

Detailed Steps

1. From the Launchpad, open the **Control Panel**.
2. Click the **Indexing Manager**.



3. Select the two new indexes and click **Rebuild**.



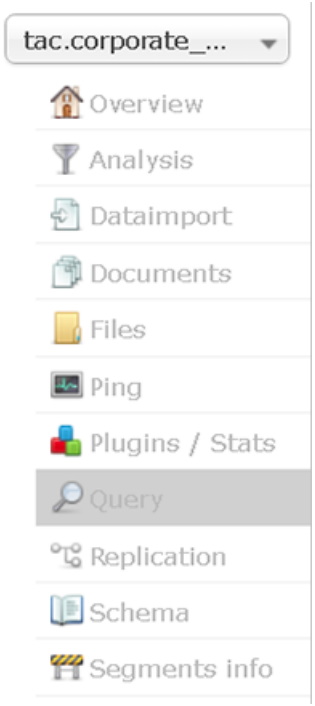
4. Close the dialog after the indexes are rebuilt. Your indexes are now ready to be used.

6. Viewing the content of the indexes

If you have done things correctly, you should now be able to log in to Solr and view the content of the indexes.

Detailed Steps

1. In a browser open the Solr administration interface (e.g. https://localhost:8983/solr/#).
2. In the Core Selector, choose **tac.corporate_events_master**.
3. Click the **Query** option.



4. Click **Execute Query** .

5. Results should appear. Notice how it is indexing the properties specified in the configuration file, in addition to common item properties (template, path, language, etc.).


```
"response":{"numFound":4,"start":0,"docs":[
    {
      "contentintro_t":"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris congue vulputate erat hendrerit tincidunt.",
      "contentintro_t_en":"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris congue vulputate erat hendrerit tincidunt.",
      "_parent":"9ee9fa33a91741bc9254056d14935c0e",
      "_name":"Hiking Pikes Peak",
      "_latestversion":true,
      "_indexname":"events_master_index",
      "_updated":"2017-08-15T12:06:28Z",
      "_version":"1",
      "_database":"master",
      "_datasource":"sitecore",
      "_group":"acecfb9d4d3b4ff38551e6afb2e2f242",
      "_templatename":"Event Details",
      "_creator":"",
      "_language":"en",
      "_uniqueid":"sitecore://master/{acecfb9d-4d3b-4ff3-8551-e6afb2e2f242}?lang=en&ver=1&ndx=events_master_index",
      "_template":"45661a37619b48abb84f5f744b267c1b",
      "_created":"2017-08-15T12:06:25Z",
      "startdate_tdt":"2017-08-16T07:00:00Z",
      "contentheading_t":"Hiking Pikes Peak",
      "_fullpath":"/sitecore/content/events/home/events/hiking/hiking pikes peak",
      "_editor":"sitecore\\admin",
      "contentheading_t_en":"Hiking Pikes Peak",
      "_version_":1575888677994758144,
      "_indextimestamp":"2017-08-16T11:55:57.719Z",
      "_path":["0de95ae441ab4d019eb067441b7c2450",
        "11111111111111111111111111111111",
        "55bb67aa167949e399b4e5a927790a8a",
        "6078f08649e64d97bce5ff084e092464",
        "9ee9fa33a91741bc9254056d14935c0e",
        "acecfb9d4d3b4ff38551e6afb2e2f242",
        "ed50d1c48abd45e2ab9eda618496c341"]},
```

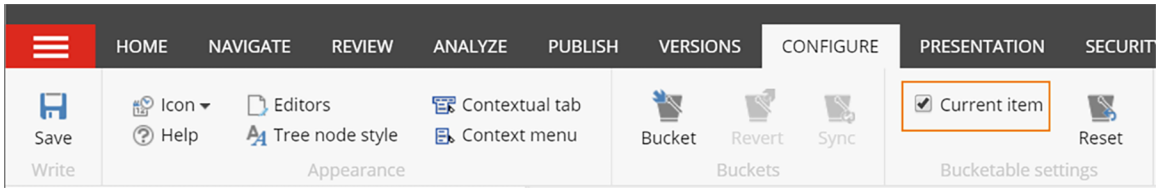

STOP HERE

Lab: Creating a Bucket

In this lab, you will convert Events List items into buckets capable of storing many Event Details items. Since you want all items of type Events List to be buckets, this setting is applied on the template's Standard Values item. Event Details items will have to be made bucketable.

Detailed Steps

1. In the **Content Editor**, open the **Standard Values** item for the Event Details template.
2. On the **Configure** tab, in the **Bucketable settings** group, select the **Current item** check box.



3. Click **OK** to confirm the operation
4. Open the **Standard Values** for the **Events List** template.
5. Click **Bucket** in the **Configure** tab.

6. Click **OK** to confirm.
7. Navigate to each of the **Events List** items (i.e. Hiking and Climbing) and click **Sync** inside the **Configure** tab.

> **NOTE:** You will need to sync the existing Events List items so their items are hidden as appropriate. New items based on the Events List template will automatically work as buckets without the need for any further action.

8. Once you have performed a Sync on a bucket, it will hide any bucketable items it contains. Expand any Events List item to see the following message: *There are hidden items in this container.*
9. If you want to see the new storage structure for the bucket items, you can select the **Buckets check box** on the **View** tab (it is deselected by default to hide bucket contents).

STOP HERE

Lab: Creating a Search-Driven Component

In this lab, you will use the Search API to render a paged list of the events within a category. This is particularly important when you are dealing with a lot of content.

The model needs to pass a list of events to the View. For each of the events, you need information about the heading, intro, image, link and date. Many of those fields can come from the index directly. You will reuse the same model representing each event as your own custom type used for searching.

Detailed Steps

1. Add references to **Sitecore.ContentSearch** and **Sitecore.ContentSearch.Linq** to your **events.tac.local** project. You can use the NuGet packages **Sitecore.ContentSearch.NoReferences** and **Sitecore.ContentSearch.Linq.NoReferences** from the NugetReferences.zip file in the Student Resources folder.
2. In the Models folder, create a new class called *EventDetails.*
3. Make the class inherit from **SearchResultItem.**

   > **NOTE:** This is not strictly necessary, but it will provide the class with some useful properties.

4. Add the following properties to the class with empty gets and sets.

Property Type	Property Name
string	ContentHeading
string	ContentIntro
DateTime	StartDate
HtmlString	EventImage

5. For the EventImage property, remove the set and override the get so it uses the **FieldRenderer** to return the correct HTMLto render the EventImage field. To get the actual item, you can use the **GetItem() method** provided by SearchResultItem.

```
return new HtmlString(FieldRenderer.Render(GetItem(), "EventImage", "DisableWebEditing=true&mw=150"));
```

```
return new HtmlString(FieldRenderer.Render(GetItem(), "EventImage", "Dis-
ableWebEditing=true&mw=150"));
```

6. In the Models folder, create a new class called *EventsList*.
7. Add the following properties to the class with empty gets and sets.

Property Type	Property Name
IEnumerable<EventDetails>	Events
int	TotalResultCount
int	PageSize

1. Creating the events provider:

You need to create a class that will perform the search and return the events using the ContentSearch API. Take care to use the correct index (Master or Web, depending on whether it is a request from the live site or the UI). The search will get all the Event Details items that are descendants of the currently requested Events List page. You need to ensure it only shows items from the context language.

Detailed Steps

1. Create a new class called *EventsProvider* in the **Business** folder.
2. Add a new method *public EventsList GetEventsList(int pageNo)*.
3. Get the index that you defined in a previous lab (events_[master|web]_index). You can inspect the current context database to establish the correct name of the index.

> **NOTE:** If you want to avoid a dependency on Sitecore.Context, which is a static class, you can also retrieve the Context item from the RenderingContext and use its Database property instead.

4. Create a new **SearchContext** and perform a search to retrieve all items of type **EventDetails**, that are a descendant of the current context item and are in the same language as the context language.
5. Use the **Page method** to get the current page requested (as passed in the parameter to the method). Assume a page size of 4.
6. Use the **GetResults extension method** to obtain the results (the list of Event Details items) and the total number of results.

7. Ensure that the GetEvents method returns an EventsList class, and populate its properties. Your code may look similar to this:

```
public class EventsProvider
{
    private const int PageSize = 4;
    public EventsList GetEventsList(int pageNo )
    {
        var indexname = $"events_{RenderingContext.Current.ContextItem.Database.Name.ToLower()}_index";
        var index = ContentSearchManager.GetIndex(indexname);
        using (var context = index.CreateSearchContext())
        {
            var results = context.GetQueryable<EventDetails>()
                .Where(i => i.Paths.Contains(RenderingContext.Current.ContextItem.ID))
                .Where(i => i.Language == RenderingContext.Current.ContextItem.Language.Name)
                .Page(pageNo, PageSize)
                .GetResults();
            return new EventsList()
            {
                Events = results.Hits.Select(h => h.Document).ToArray(),
                PageSize = PageSize,
                TotalResultCount = results.TotalSearchResults
            };
        }
    }
}
```

```
private const int PageSize = 4;
public EventsList GetEventsList(int pageNo)
{
        var indexname = $"events_
{RenderingContext.Current.ContextItem.Database.Name.ToLower()}_index";
        var index = ContentSearchManager.GetIndex(indexname);
        using(var context = index.CreateSearchContext())
        {
                var results = context.GetQueryable<EventDetails>()
                        .Where(i=> i.Paths.Contains(RenderingContext.Current.ContextItem.ID
                        .Where(i=> i.Language ==
RenderingContext.Current.ContextItem.Language.Name)
                        .Page(pageNo, PageSize)
                        .GetResults();
                return new EventsList()
                {
                        Events = results.Hits.Select(h => h.Document).ToArray(),
                        TotalResultCount = results.TotalSearchResults,
                        PageSize = PageSize
                };
        }
}
```

2. Creating the events list controller:

In the controller, you simply need to use the EventsProvider and pass the EventsList to the View.

Detailed Steps

1. Create a new Empty Controller named *EventsListController*
2. Add a parameter named *page* of type *int* to the Index method with a default value of **1**.
3. Add a dependency on **EventsProvider**.
4. Use the provider to create a model. Remember the page number is 0 based, but the page parameter in the Index method is 1 based.
5. Pass the model to the View. Your code will look something like this:

```
public class EventsListController : Controller
{
    private readonly EventsProvider _provider;

    public EventsListController() : this(new EventsProvider()) { }

    public EventsListController(EventsProvider provider)
    {
        _provider = provider;
    }

    public ActionResult Index(int page = 1)
    {
        return View(_provider.GetEventsList(page-1));
    }
}
```

3. Creating the View:

Finally, create the View to show the information. You will use the HTML from the templates in the Student Resources folder. The Pager is provided by a helper method in the TAC.Utils dll, which you imported earlier, because it was packaged as part of the Careers site.

Detailed Steps

1. Create a new **View** for the Index Action.
2. Locate the **Event-List.html** file in the Student Resources folder.
3. Copy the HTML from within the [Event List] comment.
4. Paste this inside the View.

5. At the top of the View, add the following references:

   @using events.tac.local.Models

   @using TAC.Utils.Helpers

   > **NOTE:** This last reference is not in your project, so it will give an error. However, it is in the bin folder of the webroot since it was installed with the Careers site package. Therefore, it will work when you deploy the files. If you want to remove the error in Visual Studio, copy the TAC.Utils.dll from the bin folder of the webroot and add it as a reference to your project.

6. Set the model as *EventsList*.
7. Remove the second ** tag** and all its content in the HTML
8. Add a **foreach loop** inside the <ol> tag to iterate the Events property of the model.
9. Replace the **static code** for properties in the model.
10. At the end of the View, add the following:

    @Html.Pager(Model.PageSize,Model.TotalResultCount)

11. Compile and deploy all the changes.

4. Binding the presentation:

This component needs to appear on all Events List pages inside the pageContent placeholder.

Detailed Steps

1. Use the **Sitecore Explorer** to locate the */sitecore/layout/Renderings/TAC/Events item.*
2. Create a new item named *EventsList* based on the Controller Rendering template.
3. Fill the **Controller** and **Controller Action** fields to point to the controller created previously.
4. **Save.**
5. In the **Sitecore Explorer**, locate the **Standard Values** for the Events List template.
6. Press **Ctrl+Enter** to open the **Presentation Details.**
7. Confirm that **Events** is set as the Layout.
8. Ensure that you use the templates from */sitecore/layout/Renderings/TAC/Events* when prompted. Add the following renderings to these placeholders:

Rendering	Placeholder
One-Column	main
Breadcrumb	pageHeader
EventsList	pageContent

9. **Save** the changes.

5. Verifying it all works

Navigate to an Events List page in Preview and notice that the content now appears. You might also publish the site and look at the page in the live site. If done correctly, the content in the live site and the Preview come from different indexes. Try making a change in the information of an Events Details item and see if it correctly displays in Preview. But it should not appear in the live site until it is published.

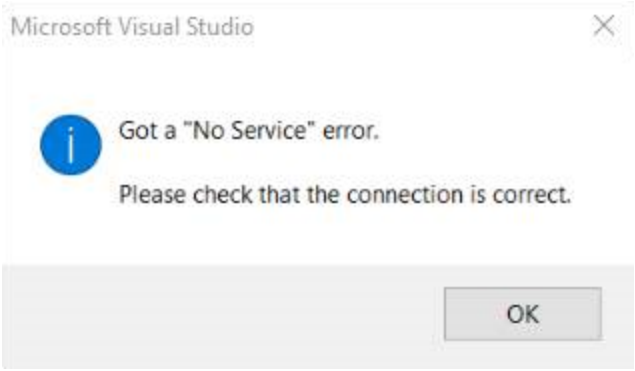
STOP HERE

Known Issues

1.0 Issue: Can't connect to Sitecore Rocks – "No Service Error"

This may occur during the lab: Setting up Sitecore Rocks, in the section: **To create a Sitecore Rocks connection, Step 3b**.

When you are testing your connection to Sitecore Rocks at this step in the Lab, you may get the following error message:



Solution:

1. In the web root of your Sitecore instance, in the Bin folder, check to see that the Sitecore.Rocks.Server.dll version is 2.1.0.79.
2. If you do not have this version of Sitecore Rocks, you can download the correct version from Visual Studio Marketplace or GitHub using the AppVeyor option.
3. If you are connected to the correct version and still can't connect to Sitecore Rocks, you need to change access restrictions.
4. You change the access restrictions in the web.config of your Sitecore instance to allow access, by adding the following code:

```
<location path="sitecore/shell/WebService">
   <system.web>
      <authorization>
         <allow users="?,*" />
      </authorization>
   </system.web>
</location>
```

5. You should now be able to connect your Sitecore instance with **Sitecore Rocks version 2.1.0.79**.

2.0 Issue: You have a Sitecore XM1 implementation, with Lucene as the default search provider

The labs require Solr as the default search provider. For information on how to configure Solr as the default search provider, please refer to the information on the Sitecore Documentation site: https://doc.sitecore.net.