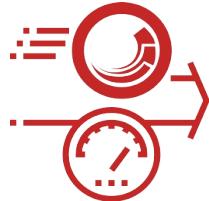


Sitecore SXA 1.8

Documentation

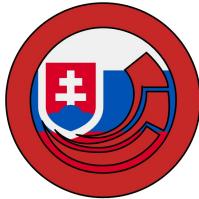
Peter Prochazka

Sitecore SXA Documentation



Compiled from [official Sitecore SXA Documentation for SXA 1.8](#)

About author:



Peter Prochazka ([@chorpo](#) / [tothecore.sk](#))

Version 1.0 / 22nd May 2019

More Sitecore guidelines and Sitecore related topics can be found on my blog [tothecore.sk](#).

You can find them also directly [in my github repositories](#).

Setting up and configuring

This section details the setup and configuration options available for SXA, select a category to see related settings:

- [Structure](#)
- [Layout](#)
- [Performance](#)
- [Rendering variants](#)
- [Search](#)
- [Security](#)
- [Sites](#)
- [Tenants](#)

Structure

The SXA content architecture includes tenants and sites. Learn more about the structuring of SXA tenant, site, and custom items:

- [SXA items in the Content Editor](#)

The SXA items in the Content Editor

SXA defines sites as Sitecore items. This topic describes the following SXA items in Sitecore:

- [The tenant items](#)
- [The site items](#)
- [The custom items](#)

[The tenant items](#)

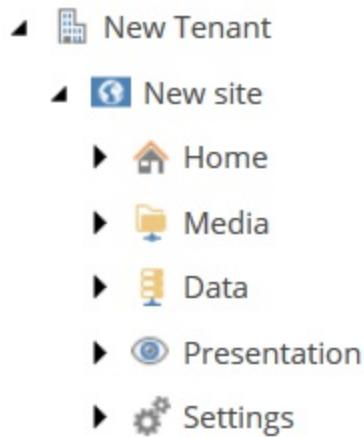
When you use the Tenant creation wizard to quickly set up your tenant and site, by default SXA creates templates, themes, and media content, with the paths stored on the tenant item you just created.

The screenshot shows the Sitecore Content Editor interface. On the left, there is a tree view of site items under 'sitecore/Content'. The 'New Tenant' item is selected and highlighted with a blue border. To the right, the 'Configuration' tab is active, displaying several configuration settings:

- Templates location [shared]:** sitecore/templates/Project/New Tenant
- Themes location [shared]:** sitecore/media library/Themes/New Tenant
- Media Library location [shared]:** sitecore/media library/Project/New Tenant
- Shared Media Library location [shared]:** sitecore/media library/Project/New Tenant/shared

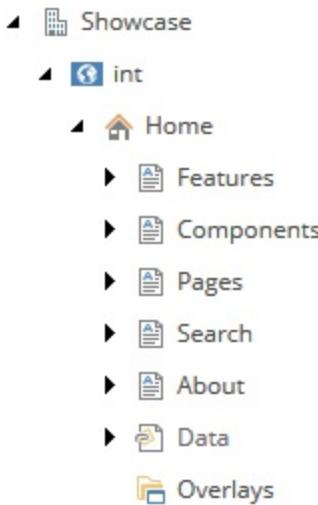
The site items

The Site creation wizard generates the structure of the site, not just a home item, but also a *Data* folder for shared content, a *Media* item for all site-specific media items, and a *Settings* and *Presentation* node for configuration and presentation details that you define at the site level.

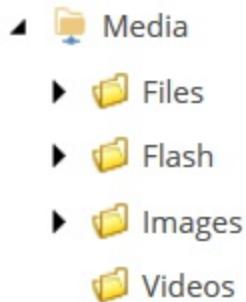


The site folder has the following subfolders:

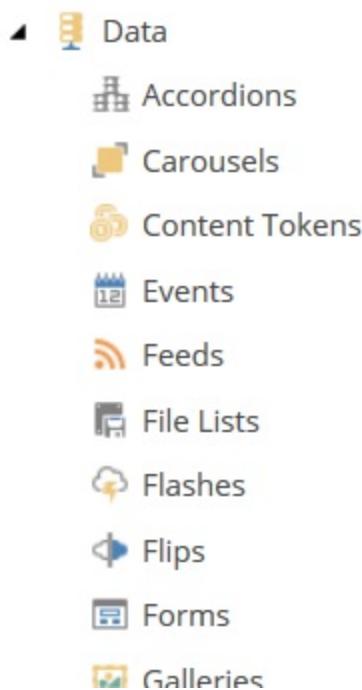
- *Home* - contains the site pages and the presentation elements that can be displayed in [overlay windows](#).



- *Media* - shows a [scoped view of the Media Library](#) that includes all the media items available for the site, such as images, animations, movies, PDFs, and so on. The media items do not exist physically in this folder but are references to the Media Library folders assigned to the site.



- *Data* - contains the data sources that can be reused across multiple pages within the website. The *Data* folder provides a location for shared data source content at the site level, and organizes this content by rendering type. Only the components that you selected in the Site creation wizard will have a *Data* container folder.



- *Presentation* - contains settings and presentation details for features that you selected when you created the site.

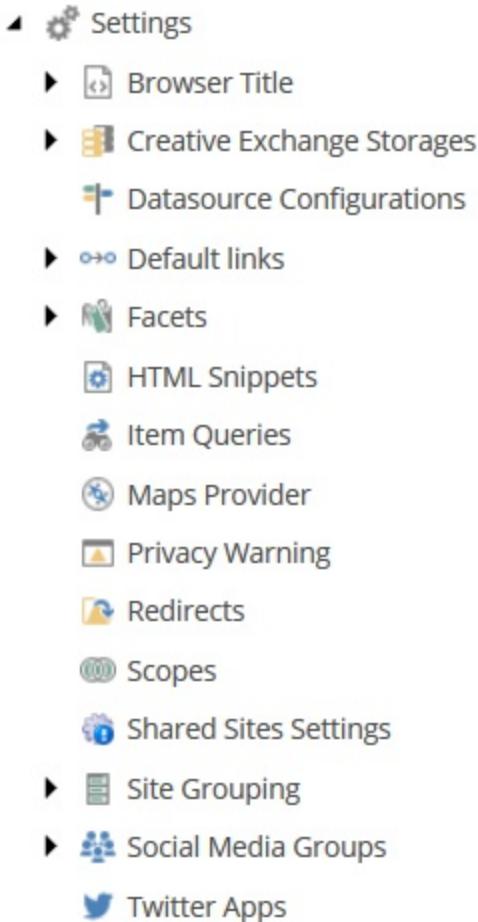
- ◀  Presentation
 - ▶  Available Renderings
 -  Cache Settings
 -  Event Types
 - ▶  Layout Service
 - ▶  Page Designs
 - ▶  Partial Designs
 -  Placeholder Settings
 - ▶  POI Types
 - ▶  Rendering Variants
 - ▶  Styles

The following table describes the items in the *Presentation* subfolder:

Item	Description
	Lists the renderings that are available to editors in the Experience Editor.
Available Renderings	SXA lets you change the structure of the components within the toolbox. This can be convenient when you want to move a component category that you use often to the top of the list or if you want to change the order of the components within the category.
Cache Settings	Set caching options for specific components . On the Home tab, click Component Cache Settings, to create your cache settings.

Event Types	Event Types is a configurable drop-down list that enables you to select available event types within event lists.
Layout Service	Specify additional JSON Rendering Variants that you can use to formulate the output of components provided by the Layout Service.
Page Designs	Create and assign page designs . This item contains the settings to change the optimization settings for a specific site.
Partial Designs	Contains your custom partial designs . Create and change a partial design by modifying this item's children.
Placeholder Settings	Set placeholder restrictions for your layout placeholders. These restrictions are applied on all pages in the site. This item is also a container for placeholder restriction items that you can assign to dynamic placeholders in Experience Editor.
POI Types	Add your points of interest .
Rendering Variants	Lists all components within your site that support variants. You can add component variants to the subfolders of this item.
Styles	SXA comes with preset styles for renderings. You can add, edit, and delete styles . In the Style section of each style you can add the renderings that can use that style.

- *Settings* - contains configuration details.



The following table describes the items in the *Settings* subfolder:

Item	Description
Browser Title	By default, the Title field is used as the browser title. To use a different field, specify a dedicated title field here. SXA also lets you define a page title prefix and suffix for each site.
Creative Exchange	Configure Creative Exchange to specify the export and import options. You can also configure the path to store

Storages	exported sites .
Datasource Configurations	SXA lets you use your own data template with a built-in SXA component. If you insert a datasource configuration and select the component, you can change the datasource template used by that component as well as override a number of its default settings.
Default links	The link component lets you add links to for example a blog page partial design to go to the next or previous article without having to link to that article explicitly.
Facets	Add facets used by the search components.
HTML Snippets	Contains the HTML snippets. The HTML snippet is a meta component that you can add to the Metadata Partial Design.
Item Queries	Create custom queries for list components.
Maps Provider	Store the Maps authorization key
Privacy Warning	Configure a privacy warning. For example, you can specify the type of warning and set the pages where you do not want to show the privacy warning message.
Redirects	Add redirect mapping .
	Add boosting rules . SXA lets you use search scopes to

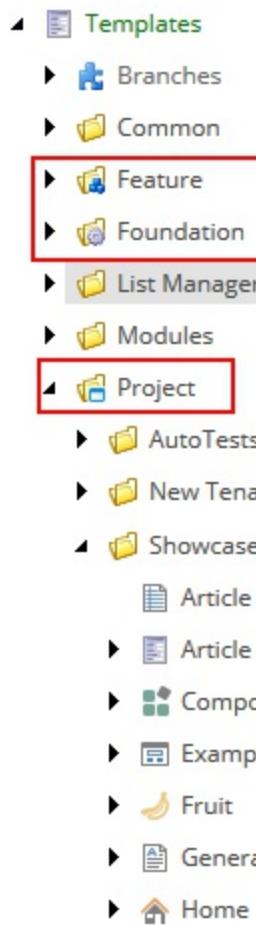
Scopes	limit search results based on conditions. Within a scope, you can boost certain results by defining boosting rules.
Shared Site Settings	View delegated areas . Do not edit this section manually.
	Children of this folder provide a Sitecore Site definition. These are equivalent to Sitecore <site> nodes that you would define within your config file if you were building the site without SXA. You can also manage these items using the SXA Site Manager .
Site Grouping	Add custom providers here. SXA lets you select different link providers for different sites. This can be useful when you have multiple sites that all have different link requirements. You can also assign a custom login page .
Social Media Groups	Define how you want to share your pages on social media. You can specify the social sharing button snippets.
Twitter Apps	Enter your Twitter app credentials. Accessing the Twitter APIs requires a set of credentials that you must pass with each request. If you have existing Twitter apps, you can view and edit your apps via the Twitter app dashboard, if you are logged in to your Twitter account on developer.twitter.com.

The custom items

When [extending SXA](#), you must think about where to store your custom items. SXA overwrites the standard SXA sections during SXA updates. Therefore, placing custom items in tree branches controlled by SXA can result in your instance being broken after an upgrade.

SXA follows the Sitecore Helix principles that contain development process recommendations for building, testing, extending, and maintaining Sitecore implementations. The template folders following Helix layers are defined as:

- *Feature* - contains concrete features of the solution as understood by the business owners and editors of the solution.
- *Foundation* - contains base logic shared by multiple feature modules.
- *Project* - provides the context of the solution. Whenever you create a new tenant it will automatically generate a Project structure similar to the following example:



The templates that SXA generates in the *Project* folder depend on the modules you choose in the Tenant creation wizard.

Layout

Learn how to control which renderings can be used for specific placeholders and how to use queries to determine the data source context.

- [Set placeholder restrictions](#)
- [Use a query to determine the data source context](#)
- [Run a source field report to set the data source context](#)

Set placeholder restrictions

Abstract

Control use of renderings for specific placeholders.

SXA enables content owners to construct web pages by dragging renderings from the Toolbox to the page. When you click and start dragging a rendering, the placeholders where you can drop the rendering light up in blue.

To have some control over the renderings that can be placed in a placeholder, you can set placeholder restrictions. For example, if you want the footer design to be simple, without too much information, you can restrict complex renderings.

This topic describes how to:

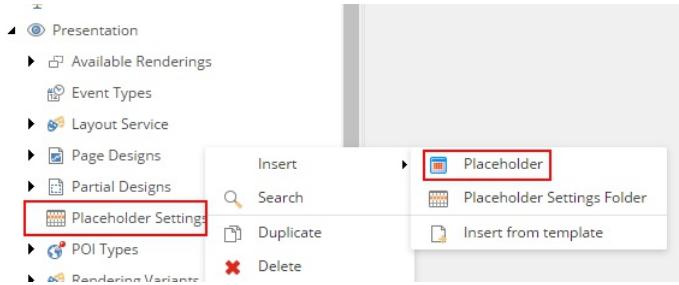
- [Set placeholder restrictions in the Content Editor](#)
- [Set placeholder restrictions in the Experience Editor](#)
- [Merge placeholder restrictions](#)

Set placeholder restrictions in the Content Editor

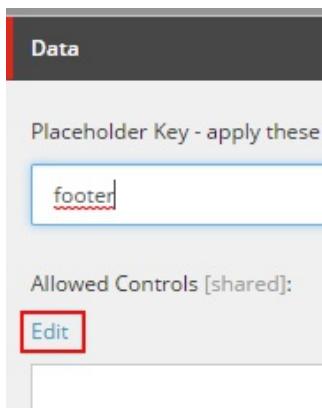
In the Content Editor, you can set placeholder restrictions for your layout placeholders. These restrictions are applied on all pages in the site. For example, if you add restrictions for a footer, these apply for all footers on every page of the site. In this way, you can control the content that content owners can place in layout placeholders.

To add a placeholder setting for a site:

1. Navigate to the Presentation folder of your site, right-click Placeholder Settings, click Insert, and then click Placeholder.



2. Enter a name and click
3. In the Data section, in the Allowed Controls field, click Edit.

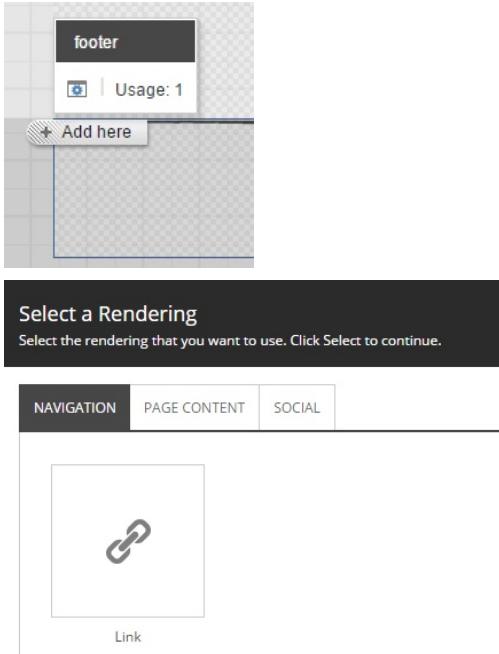


4. In the Select Items dialog box, navigate to Layout/Rendering/Feature/Experience Accelerator and select the renderings that you want to enable, and then click OK. For example, if you want a very simple footer, enable only the Rich Text (Reusable), Twitter, and Link renderings for the footer.



5. Save the placeholder setting.

Now in the Experience Editor, when you click Add here, for example, in the footer, you can only select the renderings that you just added. Other renderings can still appear in the Toolbox but the content owner is not able to drag them to the placeholder.



Note

If you add placeholder restrictions to all the placeholders of a page, then the Toolbox displays only the renderings that are allowed.

[Set placeholder restrictions in the Experience Editor](#)

When you have set a placeholder restriction for the whole site, you can select this restriction in the Experience Editor. You can also create new placeholder restrictions in the Experience Editor.

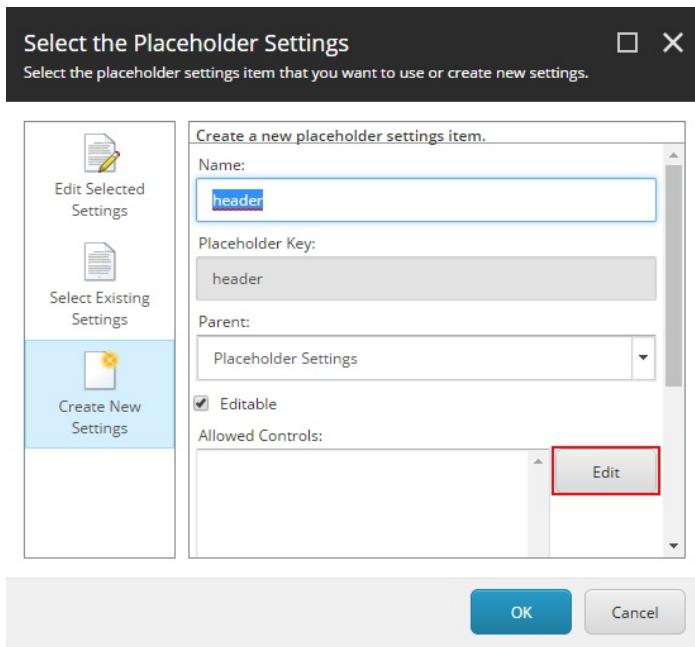
To set a new placeholder restriction in the Experience Editor:

1. In the relevant placeholder, click Edit placeholder settings.
2. In the Select the Placeholder Settings dialog box, click Create New Settings.

Note

To use an existing restriction, click Select Existing Settings, click the placeholder restriction and click OK.

3. To add renderings, click Edit and click OK.



4. In the Select Items dialog box, navigate to Layout\Rendering\Feature\Experience Accelerator, select the renderings that you want to enable, and click OK.

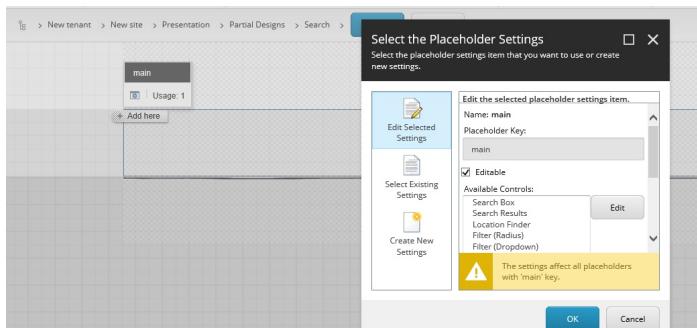
Merge placeholder restrictions

You can merge globally set placeholder restrictions with restrictions set for partial designs or specific placeholders on a page. For example, if you want to disable search renderings on all main placeholders but allow adding them on a partial design or a search page.

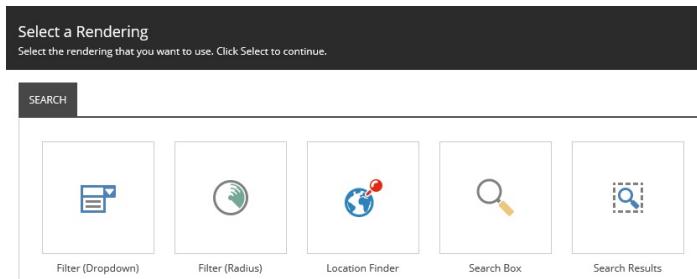
To merge placeholder restrictions:

1. To add placeholder restrictions for a partial design, open the partial design in the Experience Editor
2. In the placeholder, click Edit placeholder settings

3. In the Select the Placeholder Settings dialog box, click Create New Settings. For example, you can set a restriction for the main placeholder under (page) data.
4. Click Edit to add renderings and click OK. For example, add search renderings.

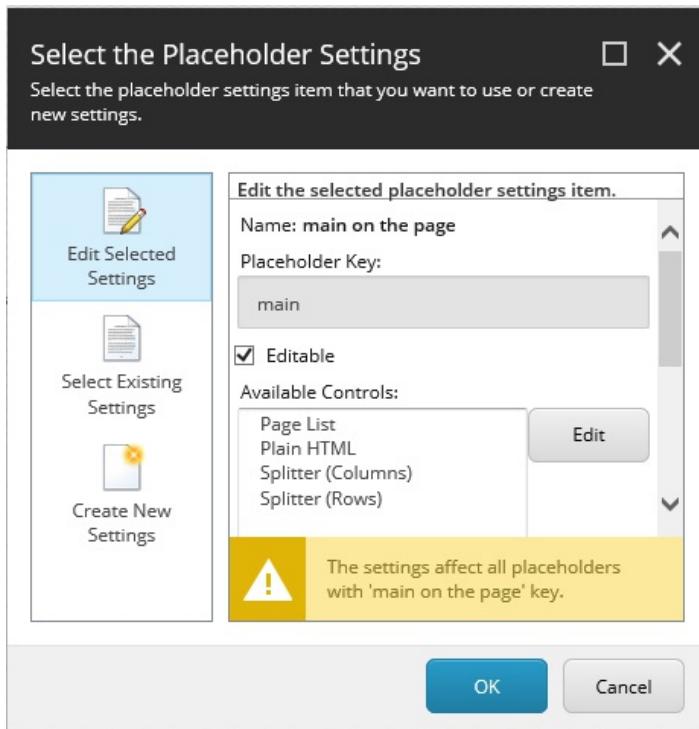


Now the partial design allows only search renderings:

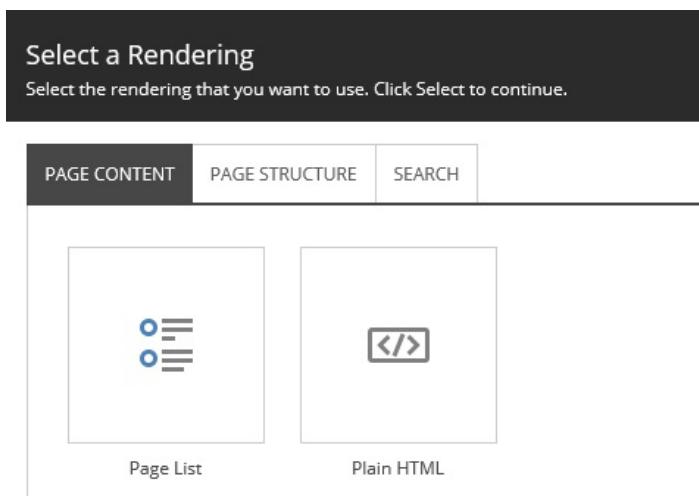


You can now merge the restrictions of the partial design with the global restrictions that are set for a page.

5. Open a page with a page design that contains the partial design that you just set restrictions for. In the main placeholder, click Edit placeholder settings and in the Select the Placeholder Settings dialog box, click Create New Settings. For example, you can set some restrictions for the main placeholder under (page) data.



6. Click Edit to add renderings, for example, page content and media renderings, and click OK.
7. Refresh the page. You can see that the settings are merged.



Use a query to determine the data source context

Abstract

Learn how to use queries for rendering variants and data sources

When you create partial designs and snippets, it can be convenient to point renderings to a specific data source. For example, if you use a partial design for three different sites and you want the renderings of this partial design to use the context of the site. You can do this by changing the data source of the renderings manually, but you can also use queries.

For example, for a shared site you use a partial design named Header that contains an image rendering. If you use a query - `query:$site/Data/Images/Logo` - in the data source field of the image, then when this partial design is used by other sites, the Image rendering tries to resolve the image data source under the current site.

You can specify a query to run on a current context item by creating a rendering variant that contains the child item `query:` This enables you to query items and use SXA tokens such as `$home` and `$site`, for example, if you want the rendering variant to display only for the child items of a certain template. You must start your query with `query` and then use standard Sitecore query syntax.

The following table illustrates examples of query syntax:

Example	Query
Children of the current page	<code>query: .//*</code>

Home item of the current site	query:\$home
Parent item of the current page	query:...
Parent of the parent item of the current page	query:.../...
Every item under site home page	query:\$home//*
Every item under site home page with additional sorting applied	query:\$home//*[@@name='News']
All items of the Page template under the current item	query:...//* [@@templatename='Page']
All items of the Page template under the Home item of the current site	query:\$site/* [@@name='Home']//* [@@templatename='Page']

This topic describes how to:

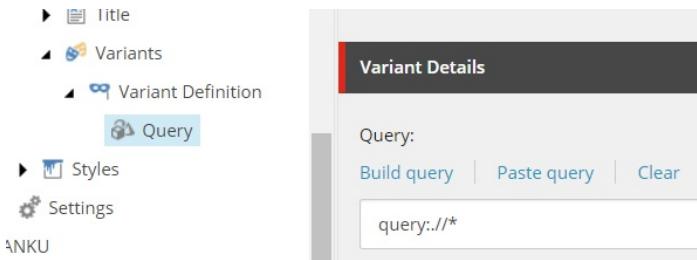
- [Add a query child item to a rendering variant](#)
- [Use a query in the data source field of a rendering](#)

Add a query child item to a rendering variant

You can use queries to switch context by adding the query child item to a rendering variant or by adding a query to the data source of a rendering.

To add the query child item to a rendering variant:

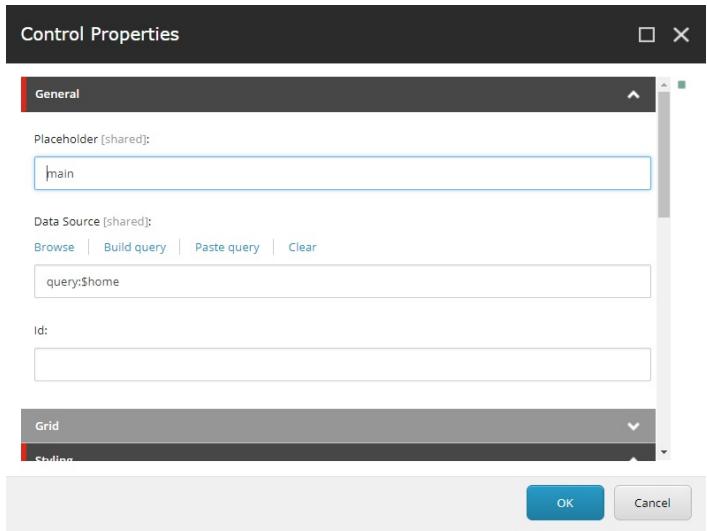
1. In the Content Editor, click the site and open the Presentation/Rendering Variants folder. This folder lists all the renderings that allow variants.
2. Right-click the rendering that you want to add the variant to, and then click Insert, Variant Definition.
3. Enter a name and the variant details.
4. Right-click the variant, click Insert, and then click Query.
5. In the Variant Details section, in the Query field enter the query.



Use a query in the data source field of a rendering

To use a query in the data source field of a rendering:

1. Click the rendering that you want to edit and in the floating toolbar, click Edit component properties.
2. In the Control Properties dialog box, in the General section, in the Data Source field enter the query.

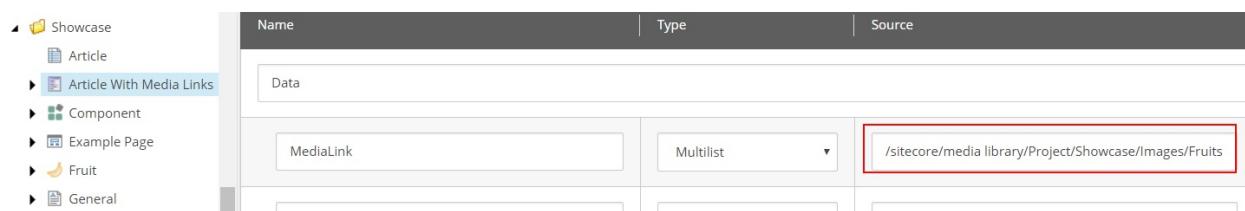


Run a source field report to set the data source context

Abstract

Use the Source field report helper script to set the data source context.

Most SXA renderings are designed for reusability and to pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. The Source field specifies where the user is allowed to look for the data source.

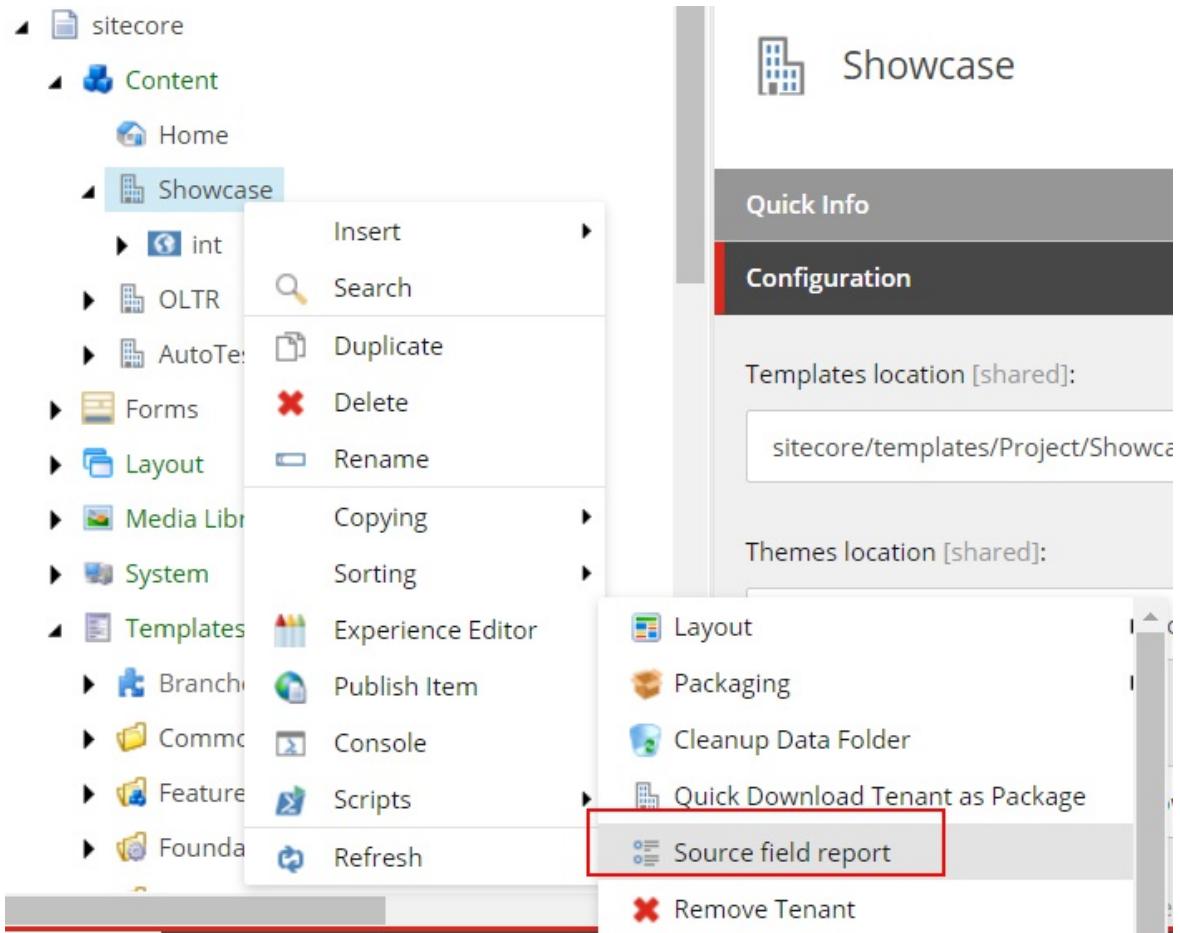


Setting this location correctly makes selecting data sources much easier because, for example, the Content Editor is directed to a specific site subfolder and does not have to search the whole Media Library for an image. You can set the location manually, use a query, or use the Source field report helper script.

To run the Source field report helper script:

1. In the Content Editor, right-click the item for which you want to invoke the Source field report. This item can be:
 - An item whose template inherits from the *Tenant* template of your tenant.
 - An item within a Project (/sitecore/templates/Project).
2. Click Scripts, and click Source field report. The script looks for the

fields that do not have a Source field specified.



3. In the Powershell Script Results dialog box, review the results. The report shows a list of all the fields that can be improved (fields with an empty Source field).

PowerShell Script Results				
Icon	Name	Path	Type	Recommendation
	Link	/sitecore/templates/Project/F/T/Product/Custom/Link	General Link	query:\$site
	Logo	/sitecore/templates/Project/F/T/Product/Custom/Logo	Image	query:\$siteMedia

4. If you think the setting is correct, click the item and click Set

recommended Source.

5. To add a different data source location manually, click the item and then click Open.

Performance

Improve the load time of your SXA web pages by setting the caching options for the SXA renderings:

- [Set SXA caching options](#)

Set SXA caching options

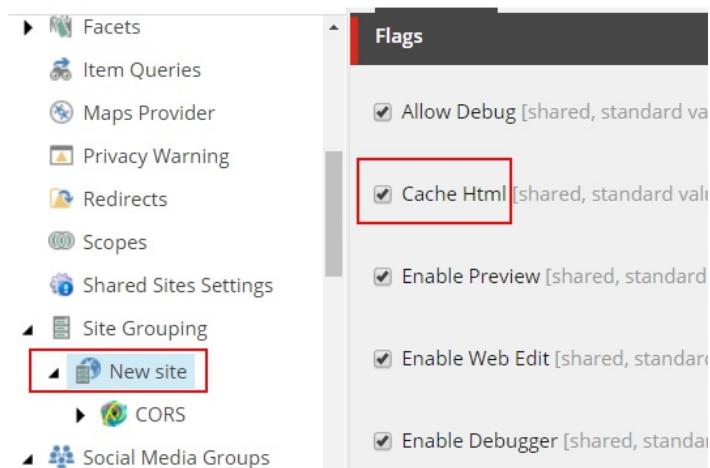
Abstract

Caching to improve performance.

You can improve the load time of your SXA web pages by setting the caching options for the SXA renderings. The SXA HTML cache minimizes the number of times Sitecore processes renderings. This is very important, especially if you have renderings that iterate over a large amount of items.

Note

To enable caching for your site, you must select the Cache Html check box. Navigate to `sitecore/content/tenant/site/settings/Site Grouping/site`, and in the Flags section, select the Cache Html check box.



Important

You can apply HTML caching on SXA items on different levels, which have different priorities. For example, the values set directly on a rendering will always take precedence over the rendering definition caching values.

This topic describes the caching levels in order of precedence, from lowest to highest, and outlines how to:

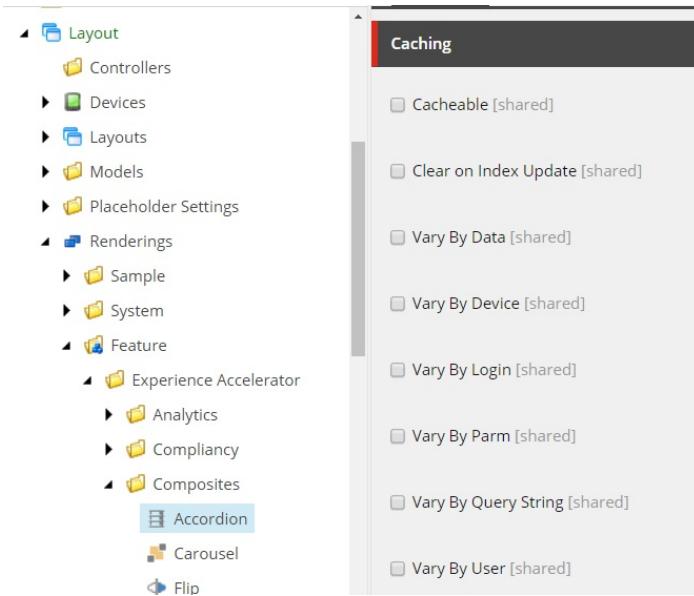
- [Set rendering definition caching options globally](#)
- [Set site level rendering caching](#)
- [Set placeholder caching](#)
- [Set specific rendering caching options](#)

Set rendering definition caching options globally

In the rendering definition item, you can specify what action Sitecore takes to render the component. If you set the cache settings in the rendering definition item, these settings are applied to all sites, unless cache settings are specified otherwise.

To set caching options for a rendering globally:

1. Navigate to sitecore/Layout/Renderings/Feature/Experience Accelerator and click the relevant rendering.
2. In the Caching section, select the caching options for the navigation rendering to help improve website performance:



- Cacheable – select to enable HTML caching. If your rendering has only one view, this is the only check box that you must select.
- Clear on Index Update – select to clear the HTML cache of renderings when the index is updated.
- Vary By Data – select to cache a separate version of the HTML based on the data source of the rendering.
- Vary By Device – select to cache copies of the output for each device that is used.
- Vary By Login – select if the rendering is displayed differently for logged in users.
- Vary By Parm – select to cache the output for each parameter accepted by the rendering. Used to distinguish different instances of the same rendering.
- Vary By Query String – select to cache the output for each unique combination of query string parameters.
- Vary By User – select if the rendering displays user-specific information.

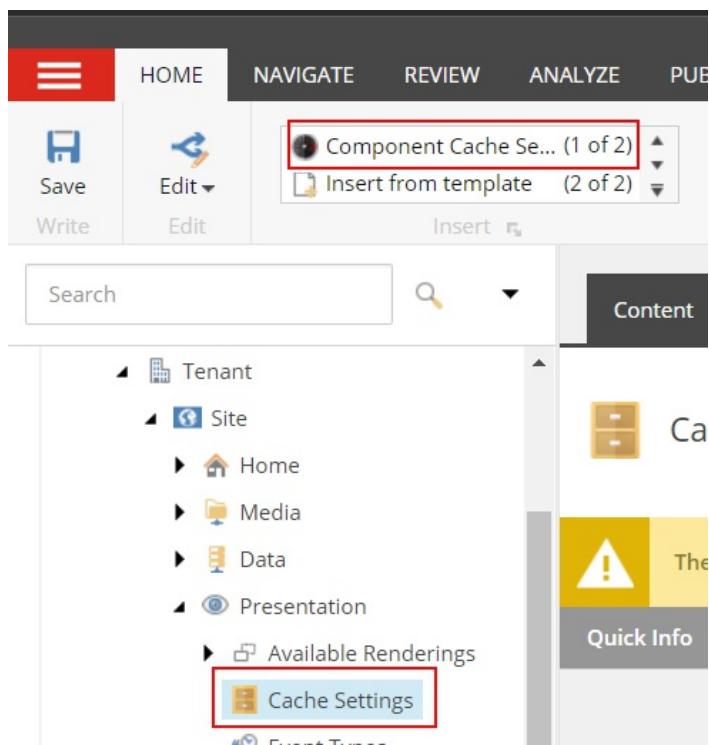
Set site level rendering caching

If you set the caching options for a rendering on a site level, this overrides the global caching settings in the rendering definition items.

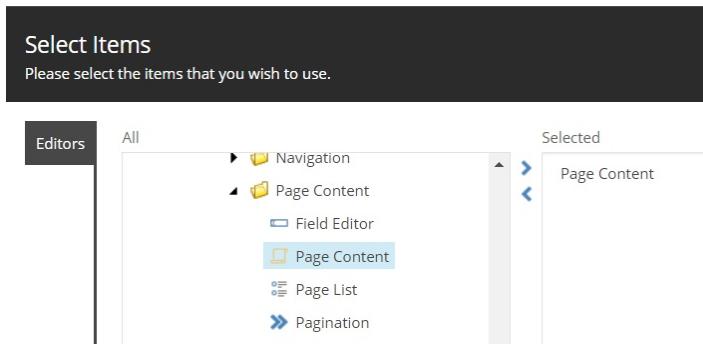
Setting caching options for specific renderings can be convenient, for example, because for a particular site you want the Page Content rendering to be cached.

To set the caching of a rendering on the site level:

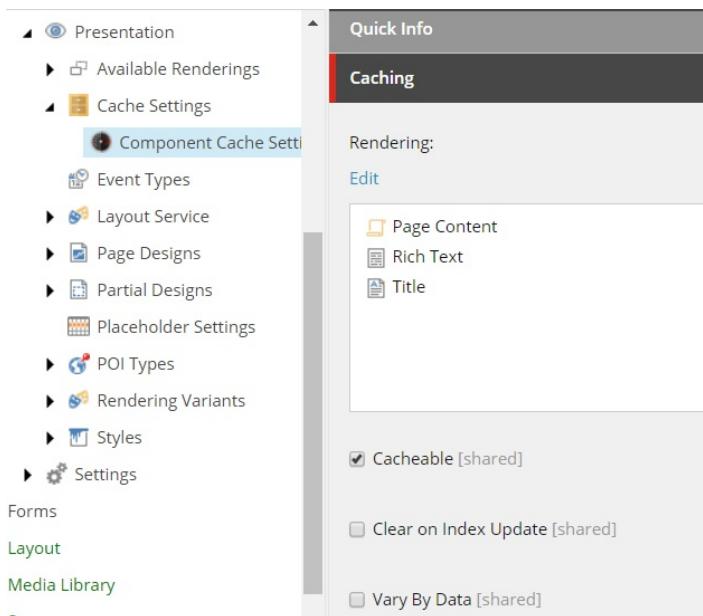
1. Navigate to `sitecore/tenant/site/Presentation/Cache Settings` and on the Home tab, click Component Cache Settings.



2. Enter a name for the setting and click OK.
3. In the Select Items dialog box, navigate to Layout/Renderings/Feature/Experience Accelerator, click the renderings for which you want this setting to apply, and use the arrow to move them to the Selected field.



4. In the Caching section, select the caching options:



5. Publish the task.

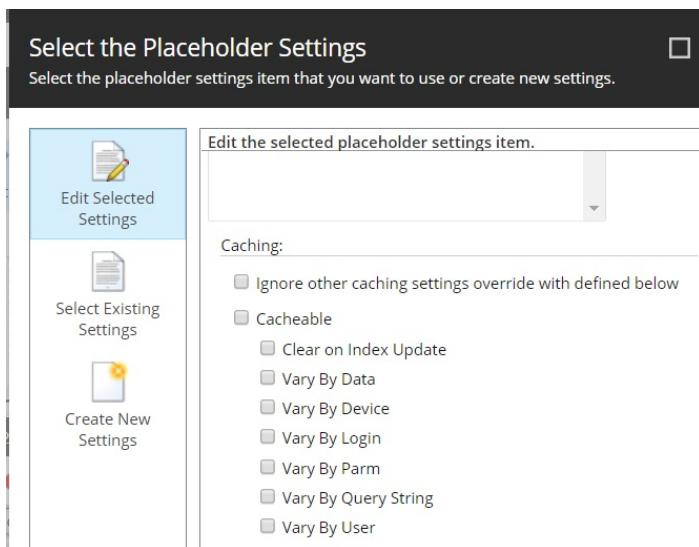
[Set placeholder caching](#)

The caching options on a placeholder override the global caching settings in the rendering definition items and the site level cache settings.

Setting the caching options on a placeholder affects all renderings in the placeholder. For example, if you use a container to add additional CSS styling to other renderings, the cache settings on the placeholder for the container applies to all renderings in the container.

To set caching options for a placeholder on a page:

1. In Experience Editor, click the placeholder, and click Edit the placeholder settings.
2. In the Select the Placeholder Settings dialog box, click Create New Settings.
3. Click Edit the placeholder settings again, and then select the caching options in the Caching section.



Set specific rendering caching options

If you set the caching options on a particular rendering, this overrides all other caching settings. This can be very convenient if your website contains pages that are the same for all users except for the user's logged in details, such as the username. If you want to cache all these pages for all the users by setting the caching options on a site level, then the entire page would be cached every time for each user with a different user name. To resolve this issue, SXA lets you apply donut caching that caches only one copy of the entire page for all the users except for a small section that remains dynamic. This section is like a hole in the cached content (much like a donut).

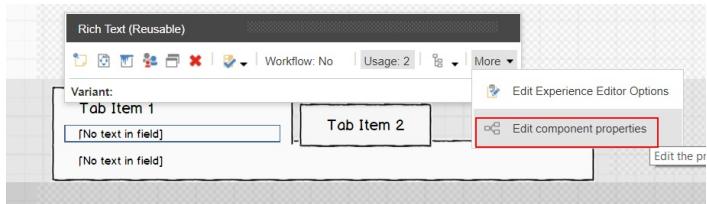
Donut caching is very useful in the scenarios where most of the pages are

rarely changed except for a few sections that dynamically change or change based on a request parameter, for example, a snippet on your page that contains renderings that pull blogs from third parties. Because you have no control over third-party websites, and the controller can fail, you do not want the output of this to be cached.

Another scenario could be a placeholder with an embedded placeholder. For example, a splitter that contains the Tabs rendering. Both the Splitter and the Tab rendering have caching applied. You can disable caching for one of the tab items to enable personalization and testing.

To set specific rendering caching options:

1. In the Experience Editor, click the relevant rendering, click More and click Edit component properties.



2. In the Control Properties dialog box, in the Caching section, select the Ignore other caching settings and the Cacheable check boxes, and then select other caching options. Alternatively, to prevent the rendering from being cached, clear the Cacheable check box.

Caching

Ignore other caching settings override with defined below [shared]

Caching [shared]:

- Cacheable
 - Clear on Index Update
 - Vary by Data
 - Vary by Device
 - Vary by Login
 - Vary by Parameters
 - Vary by Query String
 - Vary by User

Rendering variants

Rendering variants let you display different appearances of the same rendering. Learn how to:

- [Create a rendering variant](#)
- [Render image fields using the Responsive Image variant](#)
- [Enable field fallback functionality](#)
- [Walkthrough: Creating a rendering variant for a blog post](#)
- [Walkthrough: Using a rendering variant reference item to display fields](#)
- [Walkthrough: Using a rendering variant query item to render items](#)

Create a rendering variant

Abstract

Learn how to dynamically render fields in renderings.

SXA comes with a set of default renderings and rendering variants. Rendering variants are configurable adaptations of the default renderings. To encourage reusability, designers and front-end developers can also create new rendering variants. This gives authors more options in the way they present their content.

You can create your own variation of a rendering by adding a variant in the Content Editor.

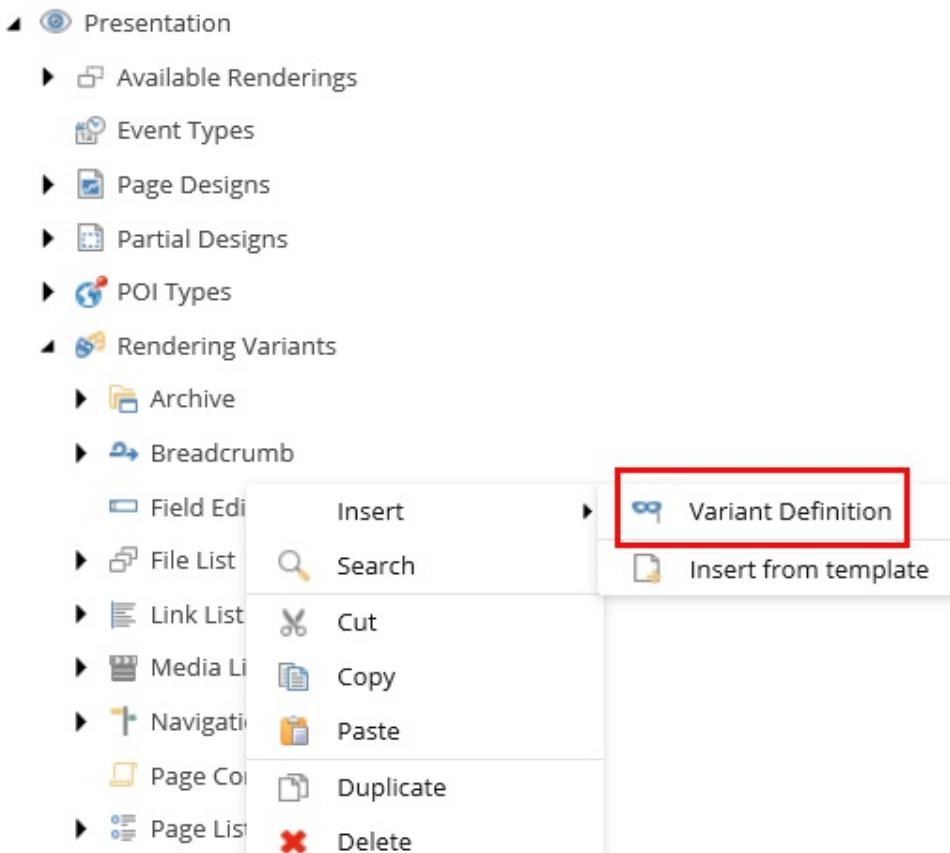
To create a rendering variant:

1. In the Content Editor, click the site and open the Presentation/Rendering Variants folder. This folder lists all the renderings that allow variants.

Note

To add a rendering to the folder, contact your Administrator.

2. Right-click the rendering that you want to add the variant to, and then click Insert, Variant Definition.



3. Enter a name and click OK.
4. In the Variant Details section, specify information in the following fields:
 - Field used as link target: provide the field name of the target item. This class is added to the list and navigation items markup (1i HTML element). This link is used to override all links when the Is link, Is prefix link, or Is suffix link check boxes are selected.
 - Allowed in templates: specify the pages that the variant is available for. Click the relevant page template, click the right arrow to move it to the list of selected items, and then click Save. If there are no templates selected, the variant is available for all pages.
 - Compatible renderings: Makes rendering variant available for other renderings. For example, if for a Page Content rendering variant, in the Compatible Renderings field, you specify Title, you can use the

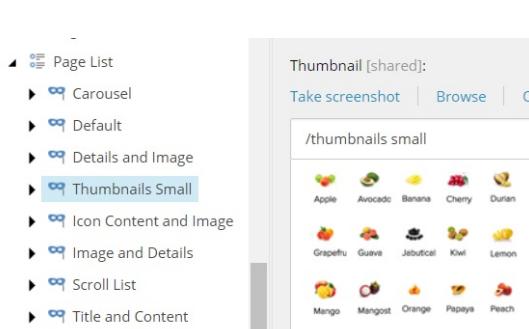
rendering variant for the Title rendering as well..

- **Css Class:** Specifies class to render into rendering content element.
For example:

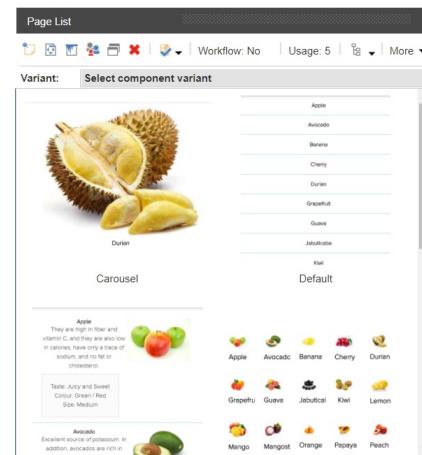
```
<div class="component title customClass col-xs-12">
    <div class="component-content">
        <div class="field-title">S</div>
    </div>
</div>
```

- **Item Css class field:** Specifies field name of the rendered item from which the CSS class will be taken and rendered into class attribute.
Supported for Navigation rendering only
5. Optionally, in the Appearance section, you can add a thumbnail image for the variant. This image will appear in the variants drop-down box and can help content editor select the best variant for their purpose.

Content Editor

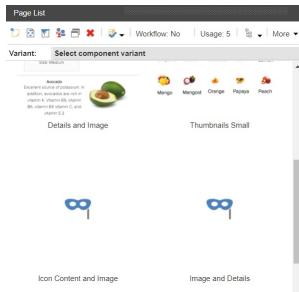


Experience Editor

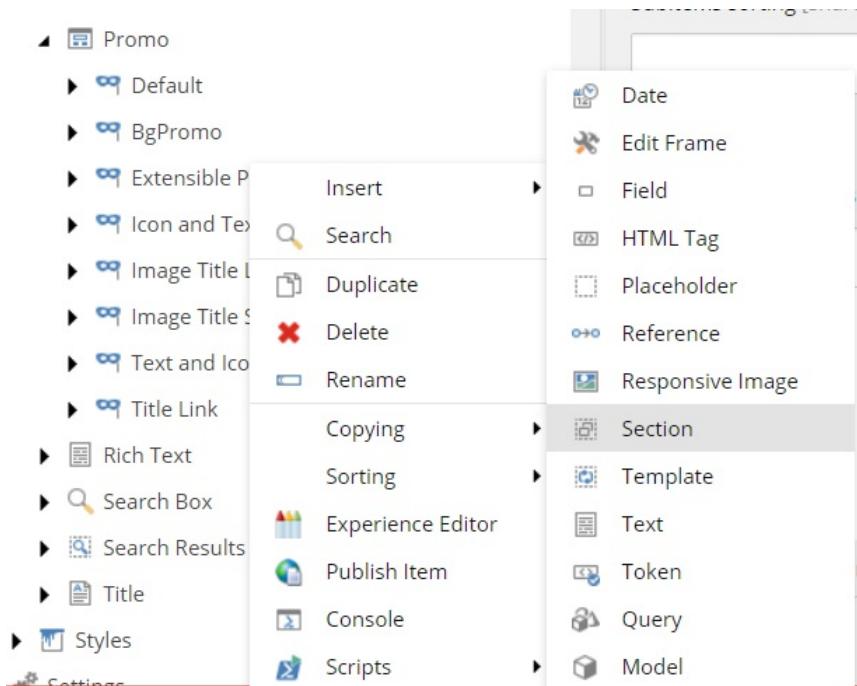


Note

Rendering variants without a thumbnail display with their name only.



- To define the fields that will be displayed, you can add child items to the rendering variant, right-click the variant, click Insert, and then click the relevant item.



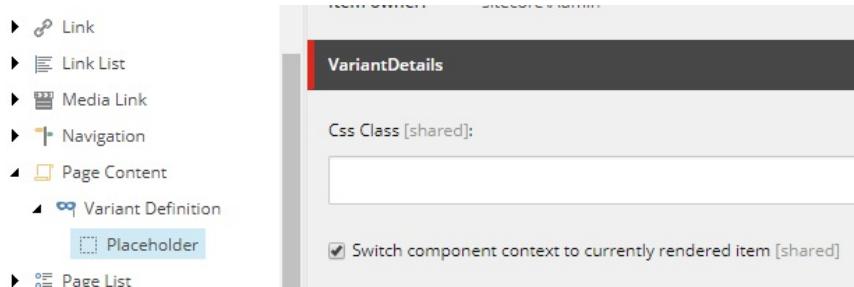
These are the child items available for a rendering variant:

- Date: displays data and time in custom format. To be able to display date and time in custom format you have to use the Date item. This item is similar to the Field item but has an additional field: "Date format" that allows you to choose a date and time format. Like the Field item, the Date item can be used as fallback item and can have its own fall back items defined.

Note

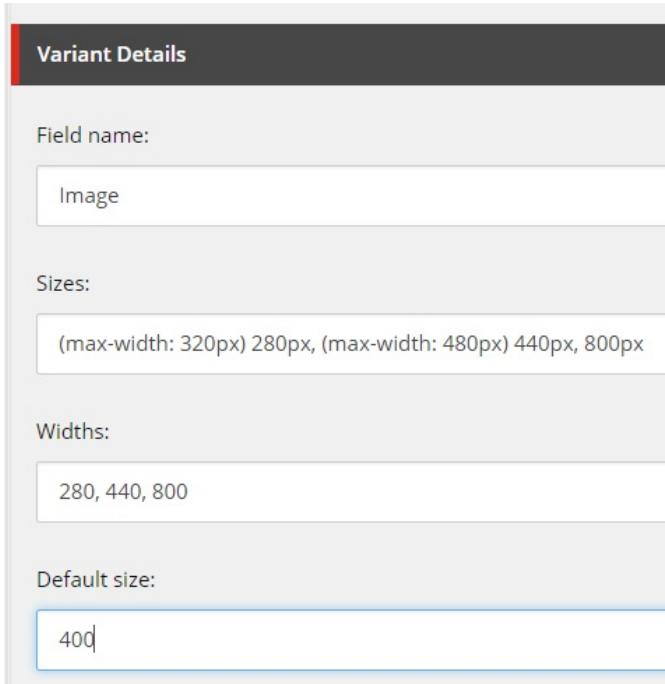
The fields and variant details are described in a table below.

- Edit Frame: variant item that can be used in the Edit mode.
- Field: displays field name and other field values.
- HTML Tag: enables users to render HTML self-closing tags.
- Model: displays properties from the current Model. The SXA renderings model inherits from `Sitecore.XA.Foundation.Mvc.Models.RenderingModelBase`. In the Property path field, you can enter the path to the property that you want to display.
- Placeholder: renders a placeholder that enables users to add additional renderings. If you want the user to be able to switch context for the renderings inside this placeholder, go to the Variant Details section and select the Switch component context to currently rendered item checkbox.



- Reference: displays field from referenced item. If you want to display a field from a referenced item, you can define this field in the PassThroughField. You can use this variant field for the following fields: LinkField (GeneralLink, DropLink), FileField (File), ImageField (Image), and ReferenceField (Droptree). Reference items can contain have another reference item as its child item. This can be convenient when you want to create a tree of reference items and display fields from items which are referenced in referenced items.
- Responsive Image: enables to define the default size as well as the

allowed sizes and width of images. Making your images responsive means that your browser serves a different sized version of that image based on the size of the image on your page and the screen resolution. The values entered in the Variant Details section will be combined in a `srcset` attribute. The values in the following screenshot will generate the following img tag:

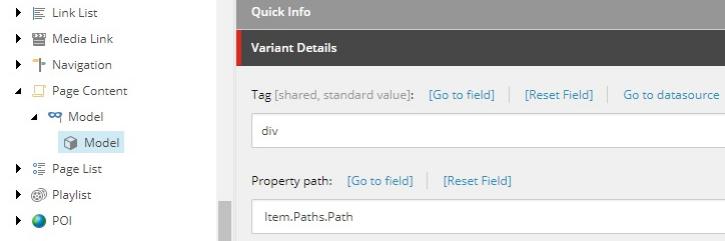


```
If left empty field content will remain unwrapped. |
| Field name       | Name of the field from the item used for rendering content.                                                          |
| Prefix           | Adds string value as a prefix.                                                                                       |
| Suffix           | Adds string value as a suffix.                                                                                       |
| Is link          | Select to have hyperlinks that wrap the field content.                                                               |
| Is prefix link   | Select to wrap prefix in the same link which you use for the field content.                                          |
| Is suffix link   | Select to wrap the suffix with a hyperlink.                                                                          |
| Is download link | Select to have a hyperlink with a download attribute.                                                                |
| Css class        | Adds Css class to the tag. If Tag field is not specified, the class will be empty.                                   |

CSS class will not be rendered.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Is editable        | Select to edit rendered field.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Date format        | Determines the date format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Render if empty    | Select to render empty element when the field is empty.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Pass through field | Defines the name of the field from the nested item.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Text               | The text to render.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Template           | <p>Define the NVelocity template that renders part of the component variant HTML. You can use the following objects:</p> <p>\$item: access to the current item (\$item.Name displays current item name).</p> <p>\$dateTool.Format(date, format): formats date and time values.</p> <p>\$numberTool.Format(number, format): formats numbers.</p> <p>\$geospatial: in case of geospatial search (\$geospatial.Distance) will show distance to certain point of interest).</p> <p>Use special tokens to format certain field values. Supported tokens are:</p> |

|                           |                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Token                     | \$Size: displays size of a file depending on size unit<br>\$FileTypeIcon: displays icon depending on file extension.                                                                                                                                                                                                                                                                             |
| Query                     | Determines the query that will be run on the current context item. For example:<br><br>To get all child items:<br><br>query: ./*<br><br>To get all items of Page template from under the current item:<br><br>query: ../../*[@@templatename='Page']<br><br>To get all items of Page template from under Home of the current site:<br><br>query:\$site/*[@@name='Home']//*[@@templatename='Page'] |
| Maximum number of results | Enter a number to limit the number of items returned.                                                                                                                                                                                                                                                                                                                                            |
| Property path             | In the Property path field, you can enter the path to the property that you want to display. For example, Item.Paths.Path will access Item property from Model, then dig deeper and access the Paths property and then the Path string.                                                                                                                                                          |



```

▼<div class="component content col-xs-12">
 ▼<div class="component-content">
 <div>/sitecore/content/F/T/S/Home/model</div>
 </div>
 </div>
 ::after
</div>

```

## DataAttributes

Adds a data attribute to the HTML element generated by the rendering variant item.

## LinkAttributes

Adds a link attribute to the link element generated by the rendering variant.

## Buttons

Edits frame buttons. Go to:/sitecore/content/Applications/WebEdit/Edit Frame Buttons to select a button.

## Placeholder Key

Defines placeholder key used for rendering placeholder.

## RenderingItem

Selects rendering used by rendering variant component.

## RenderingParameter

Specifies rendering parameters for rendering variant renderer.

Determines the default size of a responsive image. It generates responsive image syntax. For example,

## DefaultSize

```
img srcset="photo-author1-320w.jpg 320w,
```

```
photo-author1-480w.jpg 480w,
photo-author1-800w.jpg 800w"
sizes="(max-width: 320px) 280px,
 (max-width: 480px) 440px, 800px"
src="photo-author1-800w.jpg" alt="Blog auth
```

Defines media conditions for a responsive image and indicates what image size would be best to choose. to generate responsive image syntax. For example,

#### Sizes

```
img srcset="photo-author1-320w.jpg 320w,
 photo-author1-480w.jpg 480w,
 photo-author1-800w.jpg 800w"
sizes="(max-width: 320px) 280px,
 (max-width: 480px) 440px, 800px"
src="photo-author1-800w.jpg" alt="Blog auth"
```

Determines available widths for a responsive image  
Used to generate responsive image syntax. For example,

#### Width

```
img srcset="photo-author1-320w.jpg 320w,
 photo-author1-480w.jpg 480w,
 photo-author1-800w.jpg 800w"
sizes="(max-width: 320px) 280px,
 (max-width: 480px) 440px, 800px"
src="photo-author1-800w.jpg" alt="Blog auth"
```

# Render image fields using the Responsive Image variant

## Abstract

Make your images responsive.

Making your images responsive means that your browser downloads and renders a different sized version of that image based on the size of the image on your page and the screen resolution. Instead of using the standard rendering variant Field for images, you can use the Responsive Image field to add scaling and sizing criteria. With SXA, you can also use generic links to media items to render images on the page. The Responsive Image variant field lets you render the content from the media item directly. It enables you to define the default size as well as additional sizes and widths of images.

This topic describes how to:

- [Render an image field](#)
- [Render multiple images in a link field](#)

## [Render an image field](#)

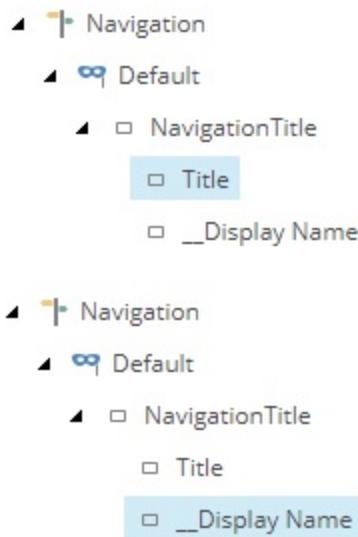
To add the Responsive Image field for scaling and sizing:

1. [Add a rendering variant.](#)
2. Right-click the variant, click Insert, and click Responsive Image.
3. In the Variant Details section, you can define the default size for your images.

The Responsive Image variant field enables you to specify:

- Field name – the name of the field that contains the link to the image.
- Sizes – define the screen widths and indicate the best image size for that screen width. For example, if you enter (max-width: 320px) 280px, (max-width: 480px) 440px, 800px this is rendered in the `<img sizes="">` attribute:

```
 Navigation
 Default
 NavigationTitle
□ Title
□ __Display Name | NavigationTitle field. |



The Title field is the first fallback item. If the NavigationTitle field does not exist or is left empty, the value from the Title field is taken.

The __Display Name field is the second fallback item. If the Title field does not exist or is left empty, the value from the __Display Name field is taken.

Note

Not all rendering variants have the same HTML hierarchy structure. For example, if you add a child item to section , this item defines new HTML that will be generated under the HTML tag selected in the section and will therefore not work as a fallback.

If the field name does not exist or is left empty, then all the child items of variant fields are fallback fields. However, if the field is editable (by default, all fields are editable), even if you have defined a fallback, in the Experience Editor, SXA will not perform fallback. Instead, you can edit the root field. This is because if the fallback was performed and the field was originally empty, there would be no way to provide the root field value. If you want fallback to be performed, you must always mark the field in the rendering variant as not editable.

Walkthrough: Creating a rendering variant for a blog post

Abstract

Quickly style a variant for blog posts.

Rendering variants let you display the same rendering in different ways. For this walkthrough, you create a rendering variant for the Page Content rendering. The Page Content rendering is a simple rendering that without variants displays a single field (like, for example, the Page Title rendering). If you add a variant, you can dynamically choose which fields to display the content.

This example uses a site that displays blog posts. You will learn how to create the variant fields and how to make sure all blog posts in your site are styled in the same way.

This walkthrough describes how to:

- [Add the template](#)
- [Create the variant](#)
- [Create the partial design](#)
- [Create the page design](#)
- [Assign the page design](#)

[Add the template](#)

This example uses a simple site with a page template that contains fields for a blog post.

Name	Type
Blog Content	
Author	Single-Line Text
WebPublication	Datetime
LeadImage	Image
Summary	Multi-Line Text

The site contains various blog posts with some data.

```

sitecore
  - Content
    - Home
    - Tenant
      - Site
        - Home
        - Blog
          - Post 1
          - Post 2
          - Post 3
          - Post 4
          - Post 5
          - Post 6
    
```

To add the template to your tenant templates folder:

1. In the Content Editor, navigate to your tenant templates folder (`sitecore/Templates/Project/`). For example, to add a custom template to your new project, go to: `/sitecore/templates/Project/projectname`.
2. Right-click the item and click Insert, New Template. Alternatively, click New Template in the Options section.

3. Enter a name for the new template, select a base template, and click Next. Templates for items that represent a page must inherit from the default *Page* template.
4. On the Builder tab, in the Add a new section field, add the relevant data template fields. For example:

Name	Type
Blog Content	
Author	Single-Line Text
WebPublication	Datetime
LeadImage	Image
Summary	Multi-Line Text

Create the variant

With the current settings, if you drag the Page Content rendering to the page, by default the rendering uses the first found variant, which in this case is the Content field.

Page Content

Variant: Select component variant

Content

© Copyright 2018, Sitecore. All Rights Reserved

To create a page content variant to display other fields:

1. In the Content Editor, navigate to Presentation/Rendering Variants, right-click Page Content, and click Insert, Variant Definition.

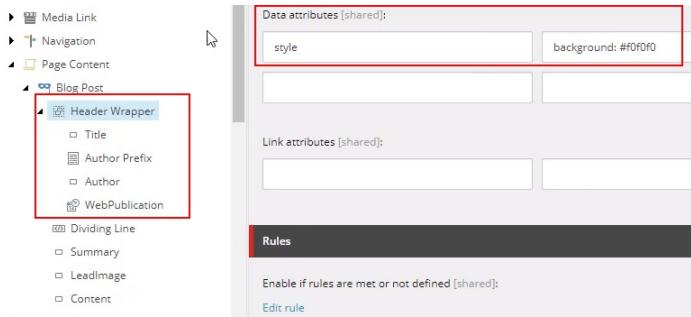
2. Enter *Blog Post* and click OK.
3. To define the fields that the rendering variant displays, you can add child items to the rendering variant. Right-click the variant, click Insert, and then click the relevant item.

For this example, lets add the following items:

Item	Name	Tag	Text
Field	Title	h1	
Field	Author	span	
Text	Author prefix	span	Author
Date	WebPublication	span	
HTML tag	Dividing line	hr	
Field	Summary	h4	
Field	LeadImage	div	
Field	Content	div	

Note

For rendering variants that have a complex structure that contains many fields, consider structuring and styling them in a section.



When you now select the Blog Post variant in the Experience Editor, the fields that you added to the rendering variant are displayed:

A screenshot of a Sitecore website. At the top, there's a header with the Sitecore logo and navigation links for 'Blog', 'Products', and 'About Us'. The main content area features a large heading 'Post 1' and a sub-heading 'Author: Adam'. Below this is a paragraph of placeholder text: 'Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit...'. Underneath the text is a dark blue banner with the word 'habitat' in white and 'Sitecore Experience Example' below it. The main text of the post reads: 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras sit amet quam lectus. Duis nec interdum ante. Aenean in mauris nec nibh viverra. Fusce fermentum massa venenatis libero malesuada, vitae dapibus risus cursus. Quisque vitae est eu enim commodo volutpat. viverra. Morbi condimentum non risus quis commodo. Pellentesque a erat sit amet metus convallis porttitor. Proin maximus magna at f pulvinar leo. Quisque efficitur ut justo sed aliquet. Praesent cursus nulla dignissim, dapibus sem eget, posuere nibh. Curabitur nulla felis purus. In id metus sit amet nulla sodales bibendum nec a turpis.' Another paragraph follows: 'Mauris ut tortor gravida, auctor leo vitae, eleifend diam. Praesent hendrerit varius augue, sit amet laoreet felis malesuada vel. Suspendisse facilisis, non tincidunt velit posuere. Cras rutrum nisl erat, egestas ornare ex vestibulum at. Donec ut iaculis est, a blandit est. Phasellus et finibus, imperdiet sapien et, tincidunt ante. Praesent ac quam ut neque dictum fringilla at vitae ligula. Nulla nisl urna, tincidunt in nul Vivamus vitae elit tincidunt, pellentesque nisl id, scelerisque nunc. Donec gravida ante a pharetra pharetra. Suspendisse placerat mauris magna, eget rhoncus leo lobortis eget. Sed consectetur eros in aliquam faucibus.' At the bottom, there's a dark footer bar with the copyright notice: '© Copyright 2018, Sitecore. All Rights Reserved'.

Note

You could decide to add styling to the rendering items. If you do this, be aware that the theme you use for your site can overwrite this styling.

Create the partial design

To style all blog posts, you can work with partial and page designs. A partial design is a set of renderings or a rendering using a specific variant that you can reuse.

To create the partial design:

1. On the ribbon, on the Experience Accelerator tab, click Partial Design, and then click Insert a new Partial Design.
2. In the Insert Item dialog box, click Partial Design, enter *Blog Post Partial* and click OK.
3. Now drag the Page Content rendering to your partial design, and select the predefined style of the Blog Post variant.

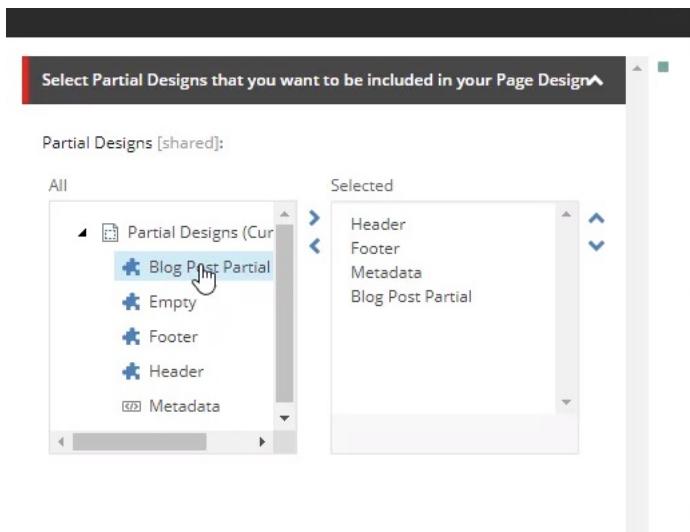


Create the page design

A page design in SXA is a selection of partial designs and renderings. A page design can contain partial designs and renderings. To make sure all blog pages in this example use the same styling, you can create a page design.

To create the page design:

1. On the ribbon, on the Experience Accelerator tab, click Page Designs, and then click Insert Page Design.
2. In the Insert Item dialog box, click Page Design, enter *Blog Post Design*, and click OK.
3. In the Select items dialog box, select the partial design(s) that you want to add. In this example, add the following: Header, Footer, Metadata, and Blog Post Partial that you just created. Click the right arrow to move them to the list of selected items, and then click OK.



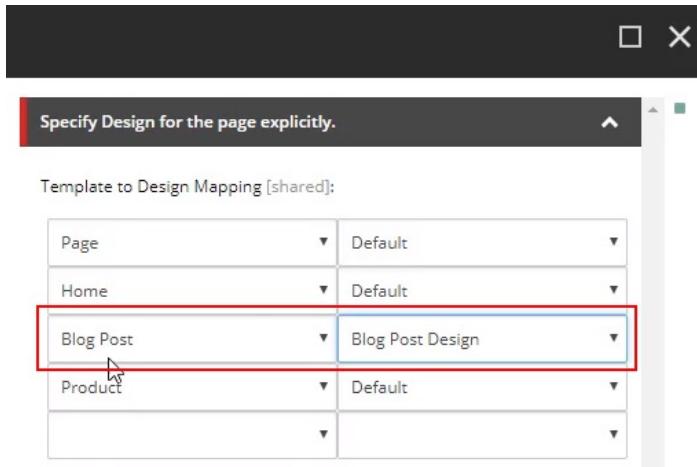
Assign the page design

Now you can make sure that all the blog pages use the same styling by assigning the page design. You can use page designs to map content types to your page layouts. By doing this, you keep the layout of your site consistent.

To assign the page design:

1. On the ribbon, on the Experience Accelerator tab, click Site Page Designs.
2. In the dialog box that appears, in the left column, select a template and,

to associate it to the page design, in the right column, select the page design.



3. To apply the changes, click OK and then click Save.

Walkthrough: Using the rendering variant reference item to display fields from a linked item

Abstract

Pass through links to get content from items that are linked.

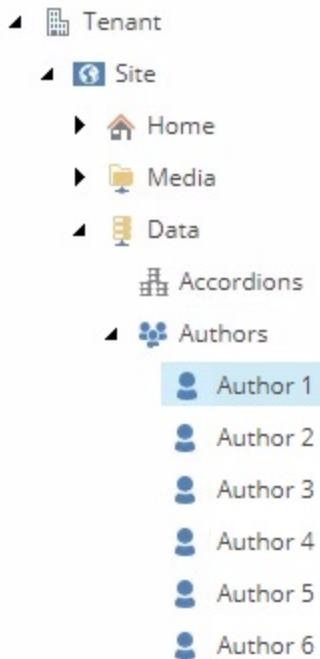
The rendering variant reference item lets you display fields from a referenced item. This walkthrough describes how to add author information to a blog post. It uses the same scenario as described in the [Walkthrough: Creating a rendering variant for a blog post](#).

This example describes how to create a rendering variant reference item that goes to the author field of the blog post, goes through the items that are linked, and renders the fields from those items.

The scenario and the end result

The scenario involves:

- A site that contains blog posts.
- A site data folder that contains a list of authors that includes their first name, last name, bio, photo, and related media items:



- A page template that refers to this data source using a query:

Project	Author	Single-Line Text	
Tenant	WebPublication	Datetime	
Author	LeadImage	Image	query:\$siteMedia
Personal Data	Summary	Multi-Line Text	
Blog Post	Authors	Multiroot Treelist	query:\$site/*[@@name='Data']/Authors/*[@@templatename='Author'] query:\$site
Home			
Page			
Page Design			
Page Design Folder			
Page Designs			
Partial Design			
Partial Design Folder			

- The blog pages have the related products selected from a list:

The screenshot shows a SharePoint navigation bar with the following structure:

- Site
- Home
- Blog
 - Post 1
 - Post 2
 - Post 3
 - Post 4

To the right, there is a sidebar titled "Selected" containing the following list:

- Author 1
- Author 2
- Author 3

The item "Author 1" is highlighted with a blue background.

The end result looks like this:

Adam Najmanowicz



Adam is a seasoned veteran in the Customer Experience Management space, involved with Content Management Systems from the very start of his career some time in the previous millennium. He enjoys devising contextually relevant content architectures that "just work" and bring a smile on the user's faces. Oh, the smile on a client's face when their "complex mission-critical issues" are resolved comes a very close second.



This walkthrough describes how to:

- [Create the rendering variant reference item](#)
- [Render the fields](#)
- [Render related media](#)

Create the rendering variant reference item

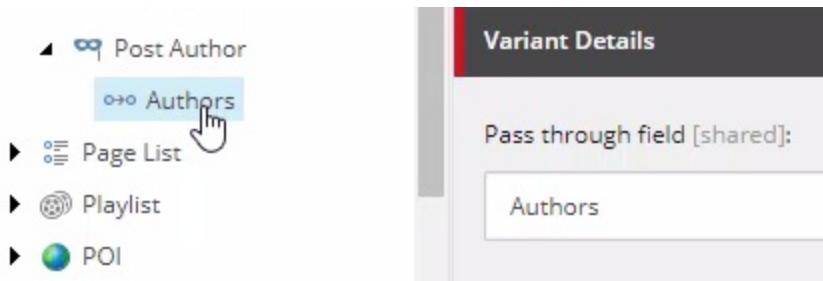
A rendering variant reference item displays a field from a referenced item. If you want to display a field from a referenced item, you can define this field in the Pass through field field.

To create a rendering variant reference item:

1. In the Content Editor, navigate to Presentation/Rendering Variants, right-click Page Content, and click Insert, Variant Definition.

2. Enter *Post Author* and click Ok.
3. Next, lets create a rendering variant reference item that goes to the author field of the blog post, and renders the content of that field.

Right-click the Post Author variant, click Insert, click Reference, enter *Authors*, and click Ok. The Pass through field field defines the name of the field from the nested item. The rendering variant now passes through the authors field.



Render the fields

Now we can start rendering fields from the authors.

To render the fields:

1. First, create a wrapper by adding a section. Right-click Post Author, click Section, enter *HeadingWrapper* , and click OK.
2. To define the fields that the rendering variant displays, you can now add child items.

Right-click HeadingWrapper, click Insert, and then click the relevant item. For this example, let's add the following items:

Item	Field name	Tag	Prefix
Field	First Name	span	

Field First Name span

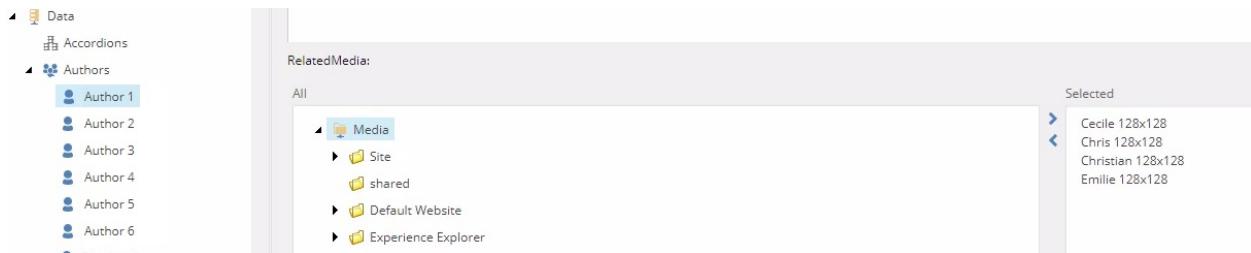
Field Last Name span

Field Picture div

Field Bio div

Render related media

As well as their picture, the authors of the sample site also have other media items attached:



To make the rendering variant render the related media associated with the author, let's use a reference item to render a responsive image item.

To render related media:

1. Right-click the Post Author variant, click Reference, enter *RelatedMedia* and click OK.
2. Right-click RelatedMedia, click Responsive Image, and click OK.
3. Empty the Field Name field.
4. In the Default size field, enter 64.

Walkthrough: Using a query to render items

Abstract

Specify a query to run on a current context item

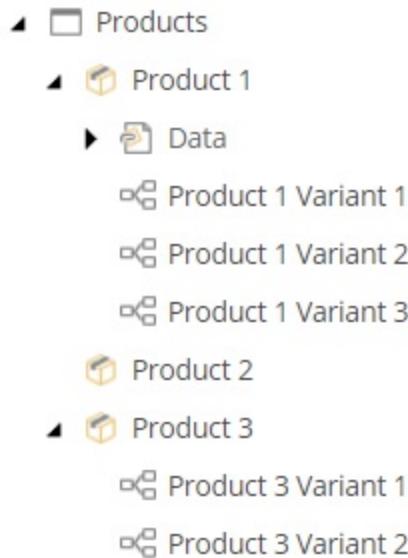
This walkthrough describes how to create a rendering variant for a blog post that renders the related products and all its child items. We will use rendering variants with a query item to accomplish this. This walkthrough also describes a more complex scenario of how to display a list of authors that share the same tag as the blog post.

The starting point for this walkthrough is the same as the scenario described in the [Creating a rendering variant for a blog post](#) walkthrough. To execute the step described for this scenario, you need an additional product list and tags.

The scenario

This scenario includes the following:

- A site that contains blog posts.
- A product list.



- A page template that refers to the product list using a query:

Name	Type	Source
Content		
Title	Single-Line Text	
Content	Rich Text	/sitecore/system/Settings/Html Editor Profiles/Rich Text XA
RelatedProducts	Treelist	<code>query:\$home/Products</code>
Image	Image	<code>query:\$siteMedia</code>

- The blog pages have the related products selected from a list:

Local

Site

- Home
- Blog
- Post 1
- Post 2
- Post 3

RelatedProducts:

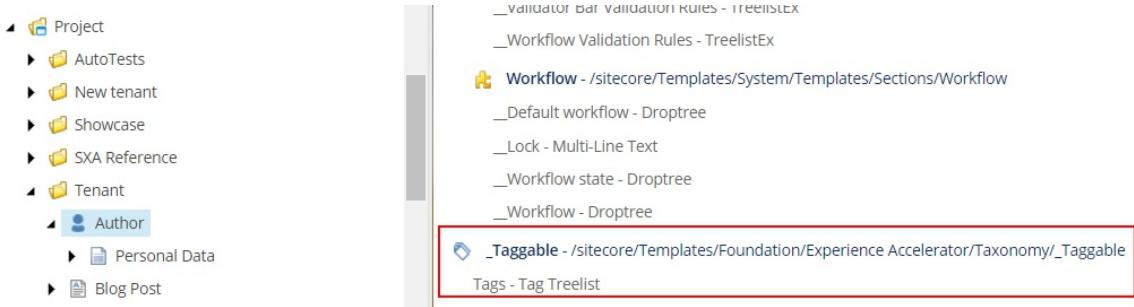
All

- Products
- Product 1
- Product 2
- Product 3

Selected

- Product 1
- Product 2

- The author template inherits the `_taggable` base template:



- The blog posts and the authors have tags assigned from a list:

The image contains two side-by-side screenshots of the Sitecore Tagging interface. Both screenshots show a 'Tags [shared]' list on the left and a 'Selected' list on the right. In the top screenshot, the 'Authors' list on the left has 'Author 8' selected, and the 'Selected' list on the right contains the tags 'Quality Assurance' and 'Management', both of which are highlighted with a red border. In the bottom screenshot, the 'Blog' list on the left has 'Post 6' selected, and the 'Selected' list on the right contains the tag 'Management'.

This walkthrough describes how to:

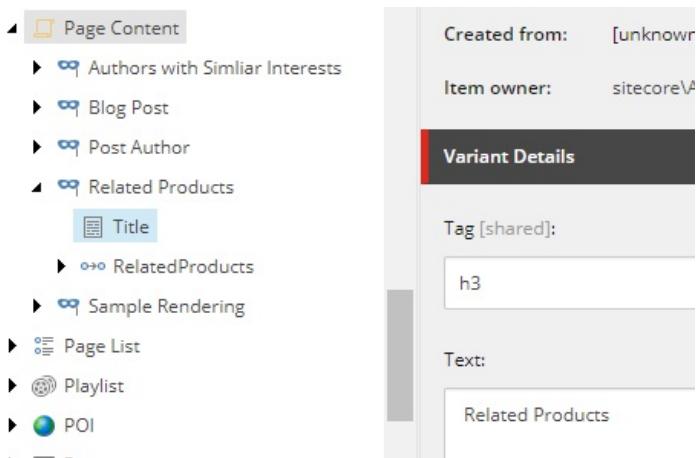
- [Add the rendering variant](#)
- [Add the reference item](#)
- [Add the query item](#)

- [Add a query item that uses a token](#)

Add the rendering variant

To be able to display a list of related products on every blog post, we will first create a rendering variant for the Page Content rendering:

1. In the Content Editor, navigate to Presentation/Rendering Variants, right-click Page Content, and click Insert, Variant Definition.
2. Enter *Related Products* and click OK.
3. Next, to create a title field, right-click the Related Products variant, click Insert, click Text, enter *Title*, and click OK.
4. In the Variant Details section, in the Tag field, select *h3* and in the Text field, enter *Related Products*.



Add the reference item

Next, we will create a rendering variant reference item that goes to the *RelatedProducts* field of the blog post, and renders the content of that field. A rendering variant reference item displays a field from a referenced item. If you want to display a field from a referenced item, you can define this field in the Pass through field field.

To create a rendering variant reference item:

1. Right-click the Related Products variant, click Insert, click Reference, enter *RelatedProducts*, and click OK.

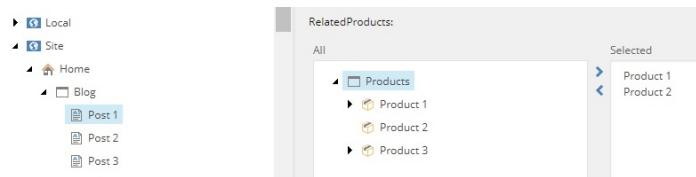
The Pass through field field defines the name of the field from the nested item. The rendering variant now passes through the Related Products field.

2. To list the products by their title, right-click the Related Products variant, click Field, enter *Title*, and click OK.
3. In the Variant Details section, in the Tag field, select *h4*.

If you now add the Page content variant to the blog page, it will look like this:



The reference passes through the *RelatedProducts* field and renders the content:



Add the query item

To be able to list the product's child items, if there are any, let's use a query.

To add the query child item to a rendering variant:

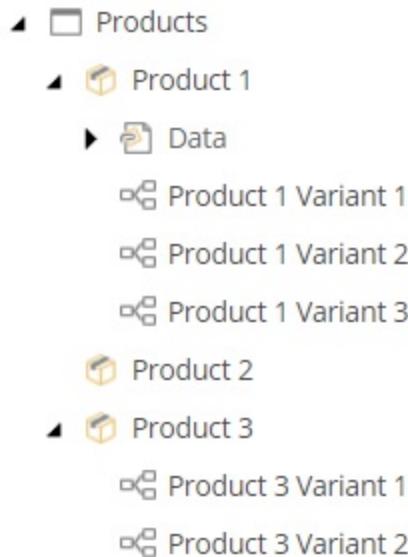
1. Right-click Related Products variant, click Query, and click OK.

2. In the Variant Details section, in the Query field, add a query that searches for all children but leaves out the title: `query: ./*
[@@templatename != 'Page Data']`

If you now add the Page content variant to the blog page, it will look like this:

The screenshot shows a 'Related Products' section with two main entries: 'Product 1' and 'Product 2'. Under 'Product 1', there are three sub-items labeled 'Product 1 Variant 1', 'Product 1 Variant 2', and 'Product 1 Variant 3'. There is no content under 'Product 2'.

The query passes through the Product list and renders the content: Product 1 with three variants and Product 2 with no variants.



3. To add a dividing line between the different products, right-click Related Products variant, click HTML Tag, enter *Line*, and click OK.
4. In the Variant Details section, in the Tag field, select *hr*.

Related Products

Product 1

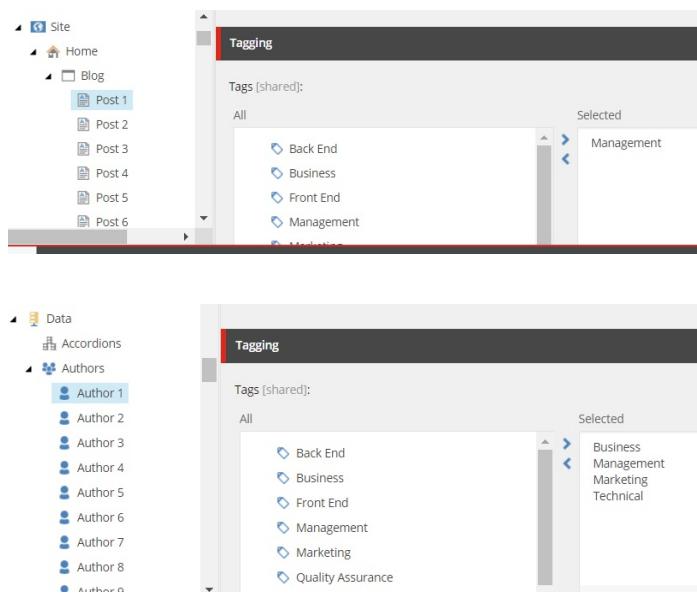
- Product 1 Variant 1
- Product 1 Variant 2
- Product 1 Variant 3

Product 2

Add a query item that uses a token

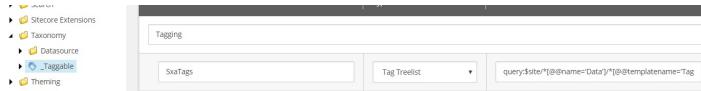
You can build more complex queries. For example, if you want to run a query on authors with similar tags.

In this scenario, the blog pages and the authors contain tags.

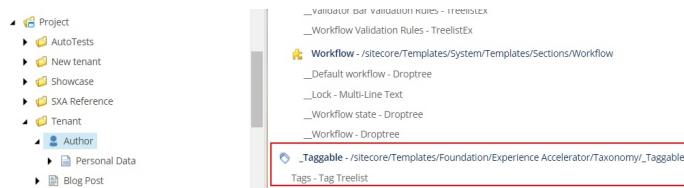


Note

If you want to execute the scenario as described in this walkthrough, you must make sure the Author template inherits the SXA _Taggable base template.



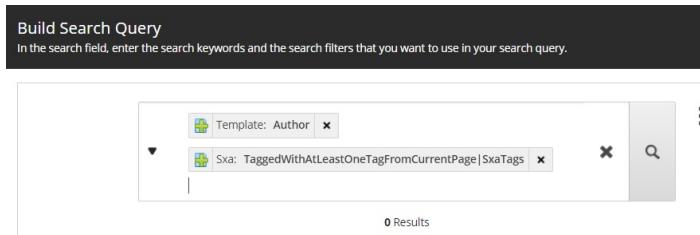
And inherits the template:



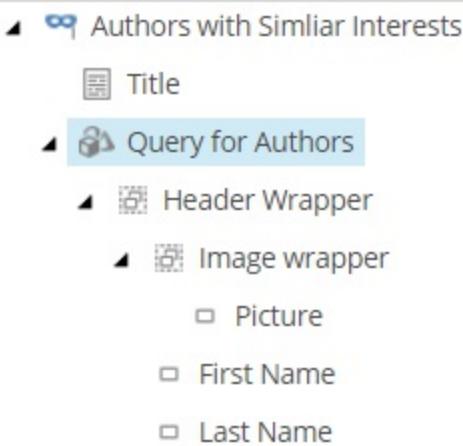
For this example, we will build a query to list all authors that have a similar tag as a blog post page.

To create a query item that uses a token:

1. In the Content Editor, navigate to Presentation/Rendering Variants, right-click Page Content, and click Insert, Variant Definition.
2. Enter *Authors with similar interest* and click OK.
3. Right-click Authors with similar interest variant, click Field, enter *Title*, and click OK.
4. Right-click Authors with similar interest variant, click Query, enter *Query for authors* and click OK.
5. In the Query field, click Build Query.
6. In the Build Search Query dialog box, first search for the Authors' template: Enter *template*, and *Authors*.
7. You can scope your search by adding the default SXA token: `TaggedWithAtLeastOneTagFromCurrentPage | SxaTags`



8. The variant also needs a Header Wrapper that contains the fields to display for the author. For example add fields for the First Name, the Last Name, and the Picture:



9. If you now add the rendering variant to blog post 1, all the authors that have the same tag as blog post 1 - *Management-* will be listed:

Authors with Similar Interests

Author 8



Author 5



Author 7



Author 1



Site

- Home
- Blog
 - Post 1
 - Post 2
 - Post 3
 - Post 4
 - Post 5
 - Post 6

Tagging

Tags [shared]:

All	Selected
Back End	
Business	
Front End	
Management	
Marketing	

Data

- Accordions
- Authors
 - Author 1
 - Author 2
 - Author 3
 - Author 4
 - Author 5
 - Author 6
 - Author 7
 - Author 8
 - Author 9

Tagging

Tags [shared]:

All	Selected
Back End	
Business	
Front End	
Management	
Marketing	
Quality Assurance	

Search

To enable visitors to quickly find what they are looking for, SXA comes with flexible out-of-the-box search functionality. When searching for a term in an SXA site, the search engine goes through all SXA items and fields in the index. You can use search scopes to limit search results based on conditions.

- [Configure SXA indexing](#)
- [Boost search results for SXA site queries](#)
- [Localize date formats for SXA search](#)
- [Integrate SXA with Sitecore Cortex™ Content Tagging](#)

Configure SXA indexing

Abstract

Define the index per site or language.

SXA lets you define index per specific language, or site. This can be convenient when you want to separate data to keep the index small and increase performance.

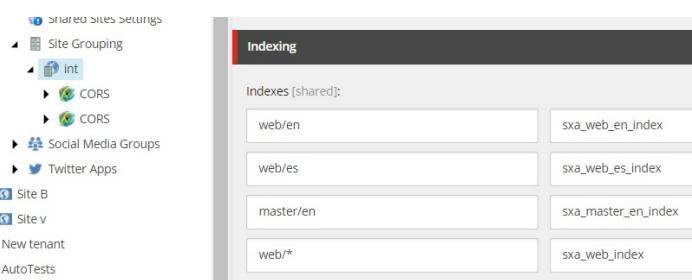
Note

To define indexing for SXA sites, you must have version 1.7.1 of SXA.

To add an index:

1. Navigate to /sitecore/content/Tenant/Site/Settings/Site Grouping/Site.
2. In the Indexing section, you can enter your index specifications for the site. Always start with the database followed by a slash. You can then specify the language or use an * to include all languages.

For example, you can create site specific indexes or you can separate between the English and Spanish language:



3. Go to your search provider config file to add the indexes you created. For example, if you use Solr, go to:
`C:\inetpub\wwwroot\Site\Website\App_Config\Include\Z.Foundation\`

and add your index.

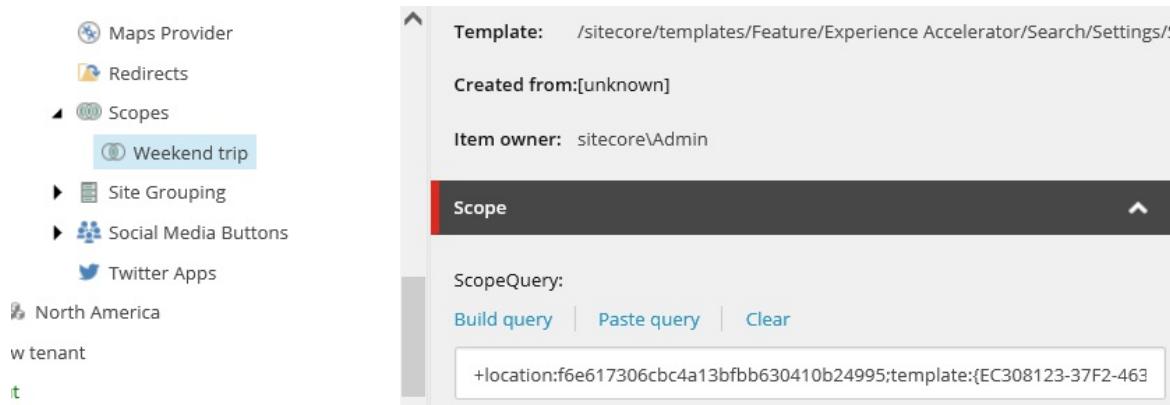
Boost search results for SXA site queries

Abstract

Boost results within a scope by defining boosting rules.

When searching for a term in an SXA site, the search engine goes through all SXA items and fields in the index. You can use search scopes to limit search results based on conditions. Within a scope, you can boost certain results by defining boosting rules. This adjusts the final score, changing the rankings of the returned results. For example, you can use boosting rules to promote the most popular books in a bookshop or to promote the sponsored pages of your website in the search results.

For a new scope, you build a search query that enables you to add several conditions. For example, for a search scope for weekend trips on a travel site, you can combine the location of the trips with the template that you use to describe the trips.



For every search scope, you can add boosting rules. For example, if for your weekend trip scope, you want to promote the trips to Amsterdam.

To create a boosting rule:

1. Navigate to /sitecore/content/Tenant/Site/Settings/Scopes/ and in the Boosting section, click Edit rule to add a new boosting rule.
2. Specify the conditions and assign the boosting values. For example, for a site that lists fruit, if you want the sour and yellow fruits to rank higher than the other fruits:

The screenshot shows the Sitecore Item Queries interface. On the left, there's a tree view with categories like Item Queries, Maps Provider, Redirects, Scopes, Components, Fruits, and others. Under Scopes, 'Sour and Yellow Fruits' is selected. The main pane has sections for 'Scope' and 'Boosting'. In the Scope section, it says 'Scope Query [shared]: template:{bda146e9-1cd3-4955-a2da-6676f677562f}'. In the Boosting section, there's a 'Rule:' section with a link to 'Edit rule'. Below it, two rules are listed: 'Rule 1' (where the Taste field contains sour, adjust boost by 50) and 'Rule 2' (where the Colour field is equal to Yellow, adjust boost by 90).

In the Create rule dialog box, there are specific predefined SXA boosting rules available. You can select a predefined condition from the list and then edit the condition. For example:

Condition

Description

Boosts the results that contain the specific field defined in the rule. For example:

Where the specific

field compares to
Search Box query

Edit rule

Rule 1

where the Taste field contains sour

adjust boost by 50

Boosts the results that contain the specific field on the specific page defined in the rule. For example:

Where the specific
field contains current
page specific field
content

Edit rule

Rule 1

where the Taste field contains current page Sour field content

adjust boost by 80

Boosts results for pages with the same tag. For example, if you use the tag sponsored for the current page, the results for pages with the same tag will be boosted.

Where the page is
tagged the same as the
current page

Edit rule

Rule 1

where the page is tagged the same as current page

Localize date formats for SXA search

SXA lets you use search renderings, such as the Filter (Date) rendering, to allow visitors to refine search results based on dates. To be able to display localized date formats, weekday names, and month names, you can download datepickers in a specific language from github and add them to your SXA site theme(s).

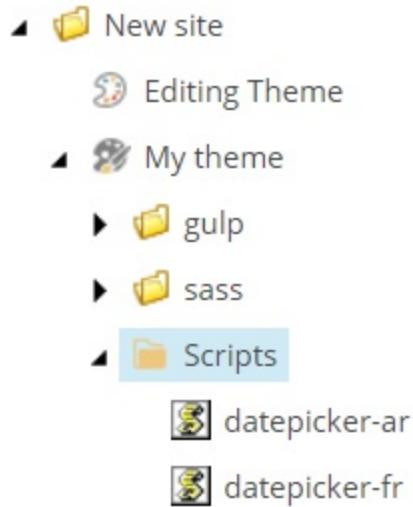
To install a local datepicker:

1. Download the relevant file from [github](#).
2. Open the file and change:

```
factory( jQuery.datepicker );  
to  
factory( $xa.datepicker );
```

3. Save the file.
4. To make the local datepicker available on your pages:

- For all themes - upload the file to /sitecore/Media Library/Base Themes/Core Libraries/scripts and position it after xaquery.
- For a specific theme - upload the file to the *Scripts* folder of that theme.



5. Switch the language of your site or page, refresh the page in Experience Editor, and go to Preview mode. You can see the calendar displays in the correct language.



Integrating SXA with Sitecore Cortex™ Content Tagging

Abstract

Use the SXA-specific taxonomy provider and tagger.

Tagging your pages is a great way to categorize them and make it easy to search and find items that include the same tags. The Sitecore Cortex™ Content Tagging feature offers an easy and accurate way to tag the people, places, companies, facts, and events that are mentioned in your content. This increases the value, accessibility, and interoperability of your content. The content tagging feature integrates the Sitecore CMS with machine learning (ML) based natural language processing (NLP) engines, such as Open Calais.

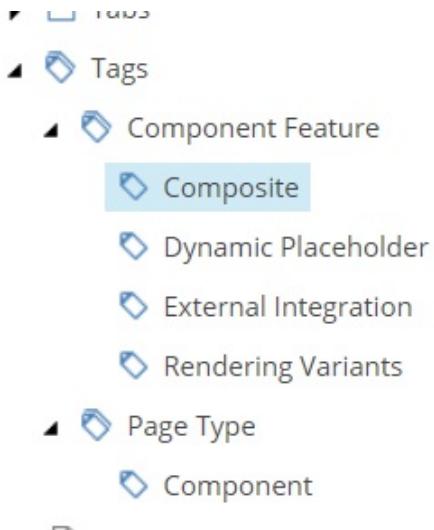
SXA integrates with the content tagging feature by providing an SXA-specific taxonomy provider and tagger.

The default Content Tagging tagger stores tags in the global Sitecore tag repository (`/sitecore/system/Settings/Buckets/TagRepository`), whereas SXA stores tags per site.

Note

To integrate SXA with the content tagging framework, you must have Sitecore version 9.1 or higher.

The `SxaTaxonomyProvider` stores tags under the site-specific tags repository: `/sitecore/content/Tenant/Site/Data/Tags`



The default Content Tagging tagger contains tag references in the Tagging section, in the `_Semantics` field (this field is only available when the Standard Fields checkbox is selected), whereas SXA uses the `Tags` field (`SxaTags`).

```
tagsList = treeView["Selected"]
compositeItem = tagsList["Composite Component"]
```

Security

This section describes how to manage security contexts and includes the following topics:

- [Set up security for a tenant and a site](#)
- [The SXA security rules](#)
- [Use a custom SXA login page](#)

Set up security for a tenant and a site

Abstract

Use the predefined SXA roles to manage user's access to items and content.

You can use the [predefined SXA roles](#) to manage user's access to items and content. Before you can assign these roles to users, you must set up security for the tenant and site.

This topic describes how to:

- [Set up security for a tenant](#)
- [Set up security for a site](#)

Note

You can only set up security for sites after you set up the security for the tenant.

[Set up security for a tenant](#)

To set up security for a tenant:

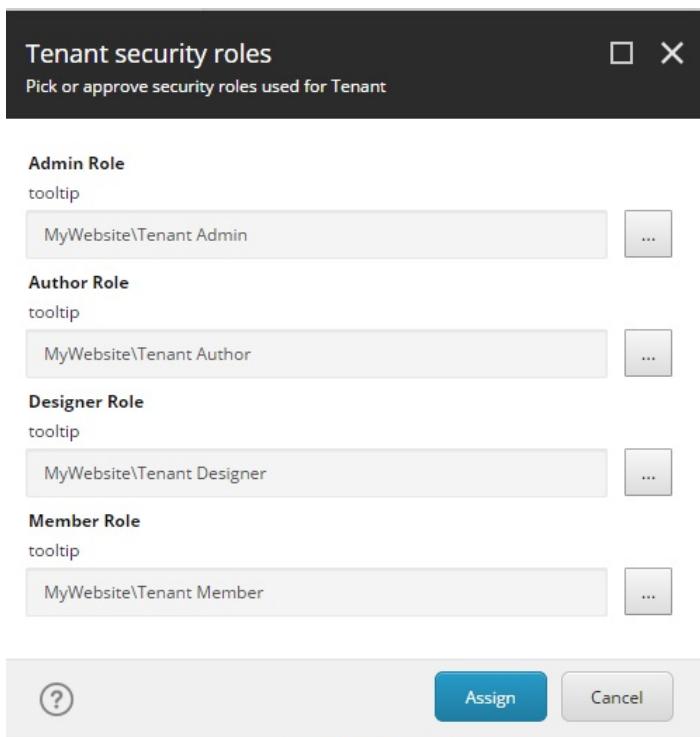
1. Right-click the tenant, click Scripts and then click SetupSecurity.
2. In the Security domain dialog box either:
 - Click the Sitecore domain, and click Proceed.
 - Click Create new domain, click Proceed and enter a domain name, and click Create.

Note

If you select a different domain than Sitecore, you must set the read/write permissions on the Languages node for one of the base roles or for the sitecore\Everyone user of your domain.

3. In the Tenant security roles dialog box:

- To add the default SXA roles, click Assign.
- To select a different role, click  and then click Assign.



Note

To clear the domain, or undo the security settings, click the Tenant and in the Security section clear the Role domain field.

[Set up security for a site](#)

Before you can set up the security for a site, you must set up security for its

tenant.

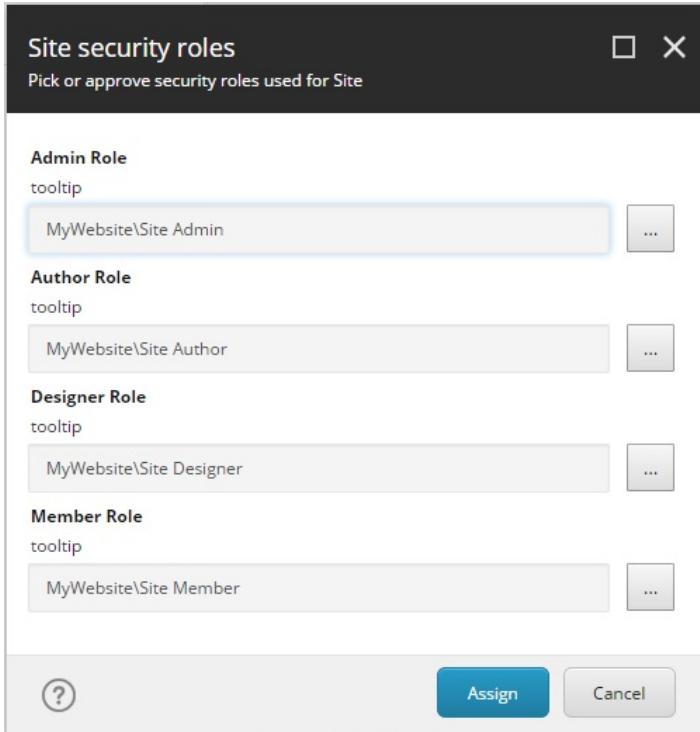
To set up security for a site:

1. Right-click the site, click Scripts and then click SetupSecurity.
2. In the Security domain dialog box either:
 - Click the Sitecore domain, and click Proceed.
 - Click Createnewdomain, then click Proceed and enter a domain name, and click Create.

Note

By default, the domain you selected for the tenant is preselected.

3. In the Site security roles dialog box:
 - To add the default roles, click Assign.
 - To select a different role, click  and then click Assign.



After you have configured security for your tenant and site, you can create new users and assign them to the Admin, Author, Designer role in the User Manager.

Note

To clear the domain, or undo the security settings, click the Site and in the Security section clear the Role domain field.

The SXA security roles

Abstract

The roles and permissions that you can set for SXA users

Security roles give your users different access rights to different areas of tenants and sites. Once you [set up security for a tenant and site](#), SXA automatically adds the SXA Admin, SXA Author, and SXA Designer roles into the SXA domain. These roles inherit Sitecore roles and are used to base the following Tenant and Site roles on:

- Tenant Member
- Tenant Author
- Tenant Designer
- Tenant Admin
- Site Member
- Site Author
- Site Designer
- Site Admin

Note

To set up security for a tenant and a site, you must have version 1.3.1 of SXA.

The following table describes the general SXA roles and their rights for working with tenants:

	Tenant Member	Tenant Author	Tenant Designer	Tenant Admin
sitecore\Content\Tenant folder\Tenant				Write (including descendants)
sitecore\Media Library\Project\Tenant				Write (including descendants)
sitecore\Media Library\Project\Tenant\Site\shared		Write for descendants		
sitecore\Media Library\Themes\Tenant\Site\ Sitecore\templates\Project\Tenant			Write for descendants	Write (including descendants)
			Write for descendants	Write (including descendants)

The following table describes the general SXA roles and their rights for working with sites:

Site Member	Site Author	Site Designer	Site Admin
sitecore\Content\Tenant			Write

folder\Tenant\Site	(including descendants)
folder\Site	
sitecore\Content\Home	Write (including descendants)
sitecore\Content\Tenant	No right to delete or rename
folder\Tenant\Site	
folder\Site\ Media	Write for descendants
sitecore\Content\Tenant	
folder\Tenant\Site	
folder\Site\Data	Write for descendants
sitecore\Content\Tenant	
folder\Tenant\Site	
folder\Site\Presentation	Write for descendants
sitecore\Content\Tenant	
folder\Tenant\Site	
folder\Site\Settings	
sitecore\Media	
Library\Project\Tenant	
folder\Tenant\Site	
folder\Site	Write for descendants
	Write (including descendants)

sitecore\Media	
Library\Themes\Tenant	
folder\Tenant\Site	Write for descendants
folder\Site	(including descendants)

sitecore\Media
Library\Themes\Tenant
folder\Tenant\Site
folder\Site\Theme

Use a custom SXA login page

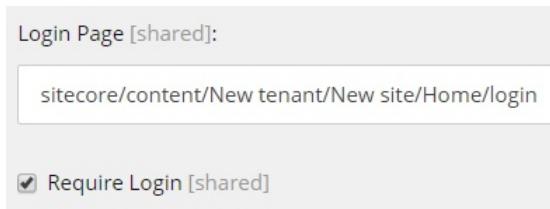
Abstract

Assign a custom login page.

If you create a custom login page for a specific SXA site, you can assign this page to be the login page. To prevent the default Sitecore controller redirecting you to the default Sitecore login page, you must use a specific security controller.

To assign a custom SXA login page:

1. Navigate to /sitecore/content/Folder/Tenant/Site/Settings/Site Grouping/Site Definition and in the Login Page field, select your login page.



2. In the Login section, select the Require Login check box.
3. To overrule the default Sitecore controller that checks if you need to be redirected to the default Sitecore login page, navigate to your login page and in the Layout section add the following:
 - In the Controller field:
`Sitecore.XA.Feature.Security.Controllers.LoginPageControl`
 - In the Controller action field: `IndexForLogin`

Login - [login]



Layout

Controller [shared]: [\[Go to field\]](#) | [\[Reset Field\]](#)

Sitecore.XA.Feature.Security.Controllers.LoginPageController, Sitecore.XA.Feature.Security

Controller action [shared]: [\[Go to field\]](#) | [\[Reset Field\]](#)

IndexForLogin

Sites

The SXA content architecture includes tenants and sites. SXA supports multitenancy, which means that you can run multiple sites on a single instance of Sitecore. Each tenant can include multiple related sites, for example, to support multiple brands for a single company or multiple languages or locations for a single brand. Learn more about configuring SXA sites by reading the following topics:

- [Add and select a custom link provider](#)
- [Add a language version to an SXA site](#)
- [Add a style for a rendering](#)
- [Clone an SXA site](#)
- [Manage multiple sites with the SXA Site Manager](#)
- [Configure a sitemap](#)
- [Configure a snippet](#)
- [Configure the map provider](#)
- [Generate a custom static error page](#)
- [Restructure the SXA toolbox](#)
- [Add a module to an existing tenant or site](#)
- [Exclude a page from the navigation menu](#)
- [Adjust link settings to enable cross-site linking](#)

Add and select a custom link provider

Abstract

How to add and select different providers for different sites.

With SXA, you can select a different link provider for different sites. This can be useful when you have multiple sites that all have different link requirements. For example, if you want the language in the URL on multilingual sites and not on a single language site. To work with a custom link provider, you can add providers to the linkManager config section.

To add and select a link provider:

1. Create a new patch file. Make sure to add your own <add/> node to the providers node under the linkManager node and add the code for your custom provider. For example:

```
<linkManager defaultProvider="sitecore">
<providers>
<add name="customLinkProvider" type="Sitecore.XA.Foundation.M
</providers>
</linkManager>
```

2. In the Content Editor, navigate to `sitecore/content/tenant/site/settings/Site Grouping/site` and in the Basic section, in the Link Provider name field, enter the name of your custom provider.

- ◀  Site Grouping
- ▶  int
-  Site A
-  Site B
-  Site C
- ▶  Social Media Groups

Language [shared]:

Link Provider name [shared]:

customLinkProvider

Add a language version to an SXA site

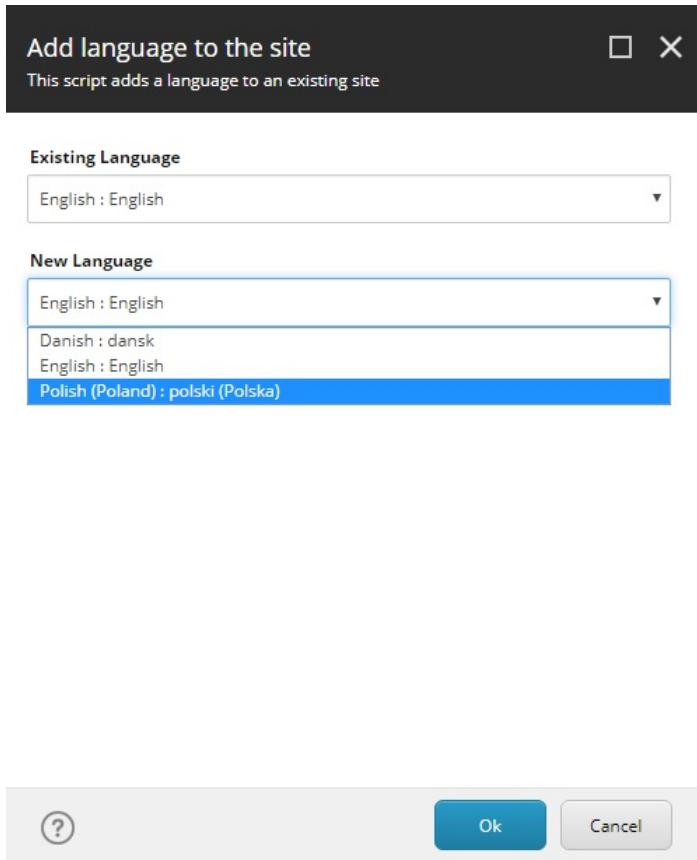
Abstract

Use a default script to deliver your SXA site in a different language version.

You can use a default script to deliver your SXA site in a different language version. The SXA language manager adds the language to all site items and copies the field values from the selected source language to a new version in the target language.

To add a language version of your site:

1. In the Content Editor, right-click the site item, click Scripts, and click Add Site Language.
2. In the Add language to the site dialog box, in the Existing Language field, enter the source language and in the New Language field, enter the target language.



3. Click OK to start the script. All site items now have new versions in the target language.

The screenshot shows the Sitecore content editor interface. The top navigation bar includes HOME, NAVIGATE, REVIEW, ANALYZE, PUBLISH, VERSIONS, CONFIGURE, PRESENTATION, SECURITY, and VIEW. The VERSIONS tab is selected. On the left, the navigation tree shows the structure: sitecore > Content > Home > Folder > Tenant > Site > Home. A search bar is present at the top. The right side displays a list of items with their language versions:

- English : English (1 version)
- Polish (Poland) : polski (Polska) (1 version)
- Danish : dansk (0 versions)

A "More languages" button is visible at the bottom of the list. Below the list, item details are shown: Item path: /sitecore/content/Folder/Tenant/Site/Home and Template: /sitecore/templates/Project/Folder/Tenant/Home -.

Add a style for a rendering

Abstract

How to add a style class item and make it available for a rendering.

SXA comes with preset styles for renderings. However, sometimes you may want to add your own custom styles. For example, you want some of the images on your site to appear without margins or if you want to add a background color to a rendering.

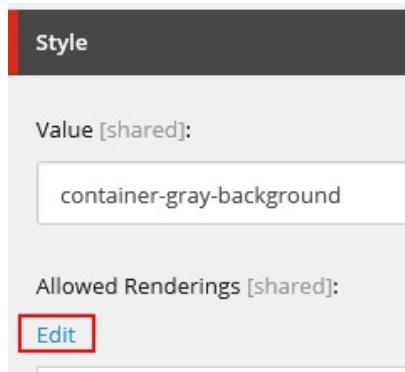
To add a class item for a rendering in the Content Editor:

1. Go to the site's Presentation folder.
2. Right-click Styles, click Insert, and then click Style to add a new style.
3. Enter the name and click OK.

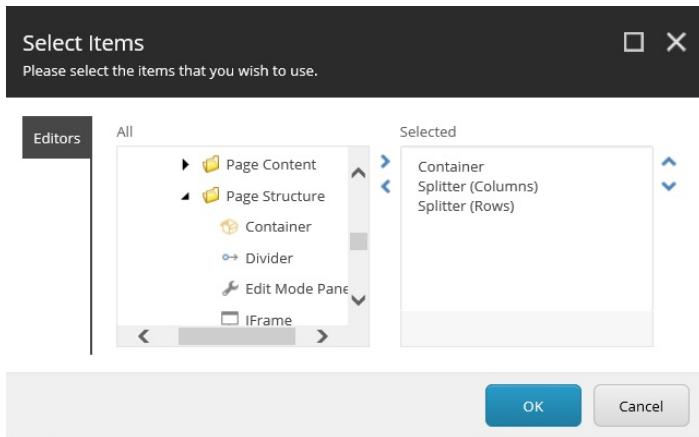
Note

Make sure that your new style has a name that helps other users to understand what it does.

4. To make the new style available for the rendering, in the Style section, click Edit.



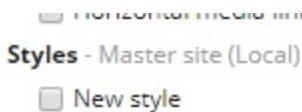
5. In the Select items dialog, click the relevant rendering, click the right arrow to move it to the list of selected items and click OK. For this example, the grey background is made available for the Container and splitter renderings.



6. To see your new style, in the Experience Editor, click edit the style.



7. In the Styles section, you can now select the new style.



8. Click OK to apply the new style to the rendering.

Clone an SXA site

Abstract

Use the clone script to copy all site content, themes, and media library items.

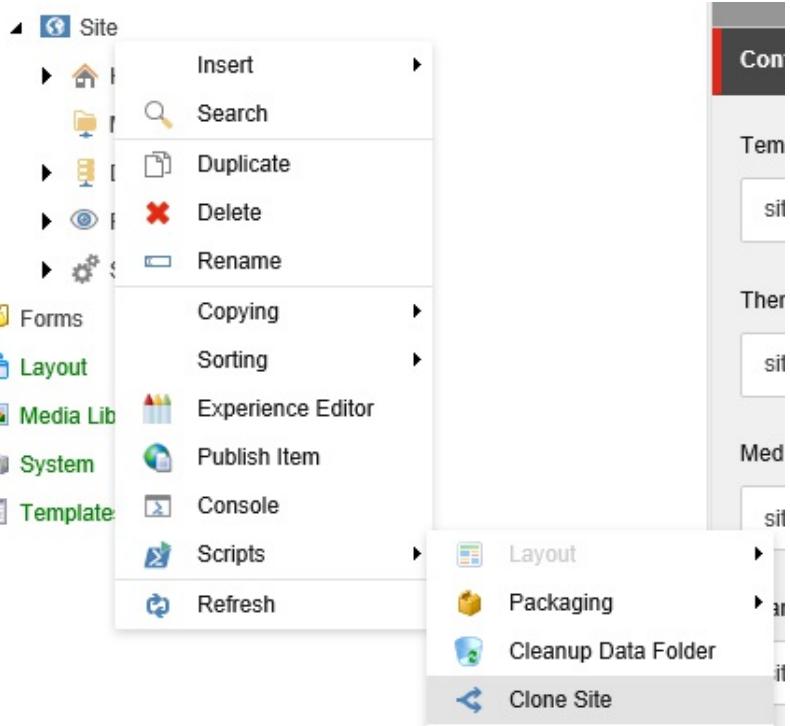
With SXA, you can quickly create a duplicate of a site by using the clone script. With a cloned site, you have an exact copy of the images, content, themes, and so on. The site operates exactly as the original, with the exception of the domain name. This can be convenient when you want to create a copy of your site to test modifications, when you want to create a live backup of your site, or when you want to deliver your site across different environments.

If you use the clone script to clone a site, the script copies all site content, themes, and media library items and changes the file locations to point to the new site, as follows:

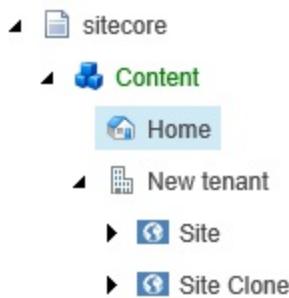
- The site content – /sitecore/content/Folder/Tenant/Site
- The site themes – /sitecore/media library/Themes/Folder/Tenant/Site
- The site media library items – /sitecore/media library/Project/Folder/Tenant/Site

To create a clone of an SXA site:

1. In the Content Editor, right-click the site item, click Scripts, and click Clone Site.



2. In the Clone a Site dialog box, enter a name and click OK. The clone of the site appears in the same Tenant item as the source site.



Manage multiple sites with the SXA Site Manager

Abstract

Resolve site conflicts.

The SXA content architecture includes tenants and sites. SXA supports multitenancy, which means that you can run multiple sites on a single instance of Sitecore. Each tenant can include multiple related sites, for example, to support multiple brands for a single company or multiple languages or locations for a single brand. By default, a new site has the * (wildcard) host mapping. If multiple sites are using the same host name and virtual folder, some of your sites may not be accessible. To solve this, you use the SXA Site Manager.

The SXA Site Manager dialog box displays a list of all SXA sites on your Sitecore instance. The icons indicate whether the site can be accessed or whether there is a conflict .

Note

The yellow icon can indicate that a site will not resolve on the current environment, but could be accessible on another environment.

If a site cannot be accessed, the Hidden domains column describes what is blocking the site. For example, in the following screenshot the Sub site A is blocked by the Master website because it uses the same host name and a virtual folder.

The screenshot shows the SXA Site Manager dialog box. At the top, there are export options (CSV, Excel, HTML, XML, JSON) and a filter field. Below the toolbar are navigation buttons for First, Previous, Last, and Next, along with a Page dropdown. The main area contains a table with the following data:

	Site Name	Status	Environment	Resolve Order	Type	Status	Conflict Details
✓	service	(Active)		*	sitecore/service	OK	
✓	modules_shell	(Active)		*	sitecore modules/shell	OK	
✓	modules_website	(Active)		*	sitecore modules/web	OK	
✓	Master_site	*	CONTENT_DELIVERY_ENVIRONMENT	*		OK	
⚠	Site_A		TESTING_ENVIRONMENT	*		Conflict	*/ hidden by */ from Master_site on * environment
⚠	Site_B			*		Conflict	*/ hidden by */ from Master_site on * environment
🚫	Site	*	(Active)	*		Conflict	*/ hidden by */ from Master_site on * environment
🚫	website					Conflict	/ hidden by */ from Master_site on * environment

At the bottom left, it says "Results: 12" and "Page: 1 / 1".

Note

In the upper-left corner of the SXA Site Manager dialog box, you can export the contents of the site overview in CSV, Excel, HTML, JSON, and XML.

By default, sites are rendered in the following order: Sitecore sites, SXA sites, EXM sites, other sites.

If your SXA sites are not resolving correctly, because of compatibility with other modules, you can mark other sites to be resolved before or after SXA sites by adding the site name and `resolve="before"` or `resolve="after"` to the `Sitecore.XA.SitesToResolveAfterSxa.config` (/App_Config/Include/z.Foundation.Overrides)

```

Sitecore Start Page Sitecore.XA.SitesTo...olveAfterSxa.config* X
1   <?xml version="1.0" encoding="utf-8" ?>
2   <!--
3
4     Purpose: This include file adds proper attribute to site definitions in order to resolve sites after SXA sites
5
6     Patch for sites definitions
7
8     To disable this file, change its extension to ".disabled".
9
10   -->
11   <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
12     <sitecore>
13       <experienceAccelerator>
14         <siteResolving>
15           <site name="mysite" resolve="before" />
16           <site name="exm" resolve="after" />
17           <site name="website" resolve="after" />
18         </siteResolving>
19       </experienceAccelerator>
20     </sitecore>
21   </configuration>
22

```

To manage multiple sites with the SXA Site Manager:

1. Click the Sitecore Start button, and from the Powershell Toolbox, click SXA Site Manager.



2. In the SXA Site Manager dialog box, to resolve the site conflicts, you can:
 - Move the site – click the site and on the ribbon, click Move Up.

The screenshot shows the SXA Site Manager interface. At the top, there are export options (CSV, Excel, HTML, XML, JSON) and a filter bar. Below the header are navigation buttons for First, Previous, Last, Next, and Page. A toolbar below the header includes Move Up, Move Down, Move First, Move Last, Open, and Publish actions.

A yellow banner at the top of the main content area states: "There are 5 conflicts in your site management configuration. Consider moving the more generic sites that hide your other sites down the list so they don't hijack the requests."

The main table lists the following sites:

	Site Name	Status	Valid for environment	Owner	Status	Notes
✓	service	(Active)		sitecore/service	OK	
✓	modules_shell	(Active)		sitecore modules/shell	OK	
✓	modules_website	(Active)		sitecore modules/web	OK	
✓	Master_site	*	(Active)	*	OK	
⚠	Site_A	CONTENT_DELIVERY_ENVIRONMENT	*		Conflict	*/ hidden by */ from Master_site on * environment
⚠	Site_B	TESTING_ENVIRONMENT	*		Conflict	*/ hidden by */ from Master_site on * environment
🚫	Site	*	(Active)	*	Conflict	*/ hidden by */ from Master_site on * environment
🚫	website		(Active)		Conflict	/ hidden by */ from Master_site on * environment

At the bottom left, it says "Results: 12 || Page: 1 / 1".

Note

To select and move multiple sites, press and hold the CTRL key when you select the sites.

- Correct the site definition – click the site and on the ribbon, click Open. In the Basic section, change the site name, environment, virtual folder, or the host name.

Note

The Valid for environment field defines where the site is resolved. Allowed values are empty string, *, or an environment name without blank spaces. The environment name is defined in the `Sitecore.XA.Foundation.Multisite.config` file.

Basic

Site Name [shared]:

Site_A

Valid for environment [shared]:

CONTENT_DELIVERY_ENVIRONMENT

TargetHostName [shared]:

Host Name (use * as wildcard and | to list more values) [shared]:

*

Virtual Folder [shared, standard value]:

/

Start Item [shared]:

3. When you have resolved all the conflicts, on the ribbon click Publish.

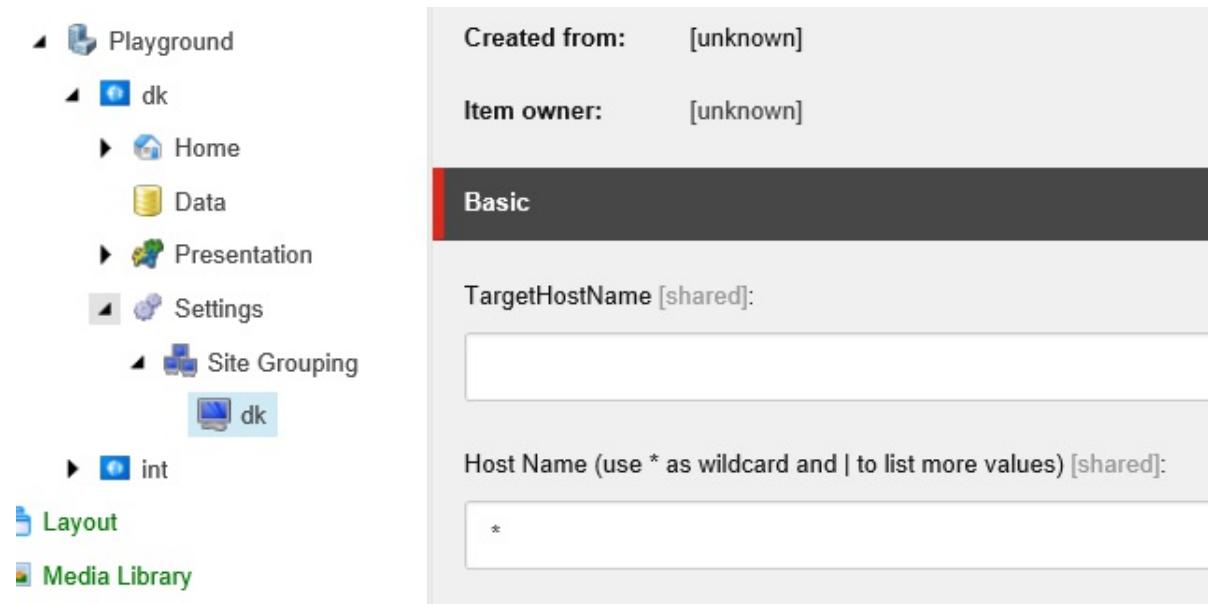
Configure a sitemap

Abstract

Set up the sitemap for a site.

Sitemaps help search engine crawlers navigate your site and improve search engine optimization (SEO). With SXA, by default the sitemap is generated for the whole site and stored in cache. An XML sitemap is created specifically for search engines to show details of the available pages in a website, their relative importance, and the frequency of content updates.

By default, the sitemap uses the Host Name defined in the Basic section in the Settings item of your site (`site/Settings`). If both the Host Name and the TargetHostName fields are empty, the sitemap returns a 404 error.



The screenshot shows the Sitecore SXA configuration interface. On the left, there is a navigation tree with items like 'Playground', 'dk', 'Home', 'Data', 'Presentation', 'Settings', 'Site Grouping', 'int', 'Layout', and 'Media Library'. The 'Settings' item is expanded, showing 'Basic' and 'Site Grouping' sub-items. On the right, there are two main configuration panels. The top panel is titled 'Basic' and contains fields for 'Created from:' (set to '[unknown]') and 'Item owner:' (set to '[unknown]'). The bottom panel is titled 'TargetHostName [shared]' and has a text input field containing an empty value. Below it is another panel titled 'Host Name (use * as wildcard and | to list more values) [shared]', which also has an empty text input field containing a single asterisk (*).

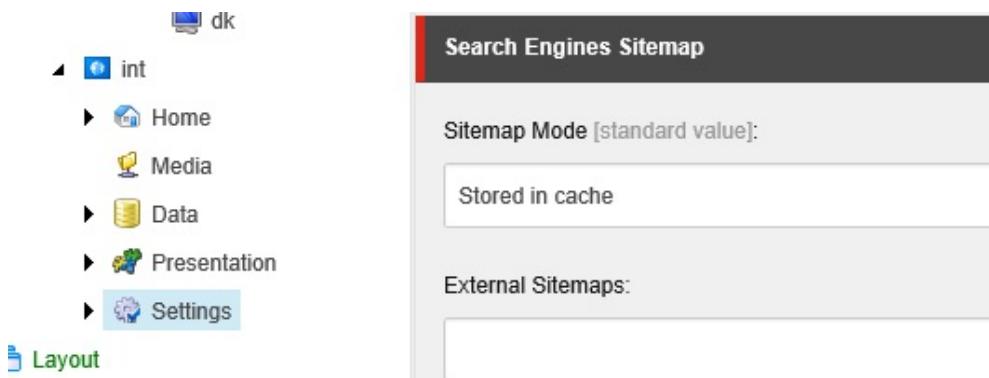
This sitemap file is stored in the root folder of your server, and is usually named `sitemap.xml`.

Every entry in the sitemap contains the following attributes:

- Loc – the location of the page.
- Lastmod – the date when the page (under loc) was last modified.
- Changefreq – how often the page changes its content.
- Priority – number between 0 and 1 that represents the importance of specific page.

To configure the sitemap for your site:

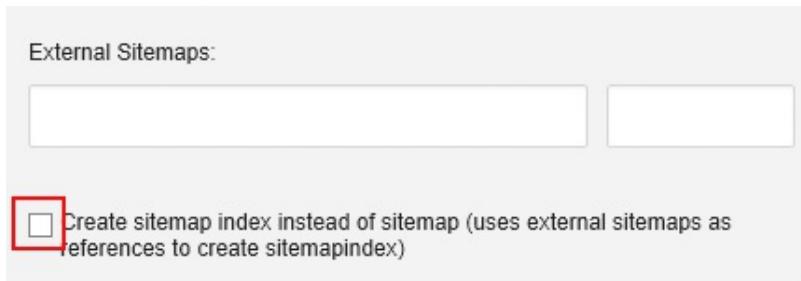
1. In the Content Editor, navigate to site/Settings and in the Search Engines Sitemap section, in the Sitemap Mode field, select the storage option for the sitemap.
 - Inactive – turns the sitemap off.
 - Stored in cache – stores the sitemap for the whole site in cache. Select this option if your site is hosted on an environment such as Azure and you cannot easily store files on a drive, or if your site is very dynamic and you need to re-generate the sitemap almost every time it is requested. This option is turned on by default.
 - Stored in file – stores the sitemap for the whole site in file. Select this option if you have a large site that does not change frequently.



Optionally, you can reference external sitemaps in the generated index file by adding the sitemap in [KEY][VALUE] format, where KEY is name of your choice and VALUE is a direct link to an external sitemap file. For

example: <http://example.com/sitemap.xml>

2. If you want to reference the external sitemaps in a sitemap index file, select the Create sitemap index instead of sitemap check box. This can be useful if you host external sites such as a WordPress blog on your site.



3. Save the settings and publish the item. Once the settings are saved, you can find the sitemap here: TargetHostName/sitemap.xml. A link to the sitemap is automatically added at the end of the robots.txt file.
4. Publish the page to automatically update the sitemap.

Configure a snippet

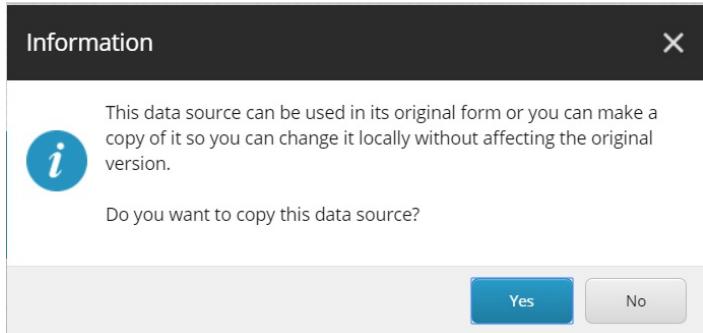
Abstract

Set how snippets can be reused.

Snippets enable content editors to work more efficiently by reusing predefined sets of renderings on site pages. You can specify the behavior of snippets reuse in the Content Editor. For example, whether you allow content editors to duplicate and change the snippet locally.

To configure a snippet:

1. Navigate to the Data folder of the site, right-click Snippet, and click Snippet Item.
2. In the Datasource configuration section, in the Global data source selection behavior field, select one of the following options:
 - Do not copy – The default setting. The snippet uses a globally assigned data source. The source of the content is in the site data folder. When you reuse this snippet on different pages and then change it, all instances of the snippet on all pages are updated automatically.
 - Copy global data source to local context upon selection – A copy of the snippet, its configuration, and content are added to the page. Changes made to the snippet are only applied for that specific page.
 - Ask user whether the copy of global data source to local context is required upon selection – This option allows the content editor to choose how to reuse the snippet and displays the following message in the Experience Editor:



Note

If you do not see the Datasource configuration section, you must add the Global Datasource Behavior template to the snippet item. To do this, navigate to the Templates folder and go to Feature/Composite/Datasource/Snippet/Snippet Item and in the Data section add the Global Datasource Behavior template (/sitecore/system/Settings/Foundation/Experience Accelerator/Local Datasources/Enums/Datasource Selection Behavior).

Configure the map provider

Abstract

Before you can use the Map rendering on your pages, you must configure the maps provider.

The Map rendering makes it easy to embed interactive maps anywhere on your page. You can also add points of interest to the map, and define a custom starting location. Before you can use the Map rendering on your pages, you must configure the maps provider. By default, SXA provides two choices of maps providers: Google Maps and Bing Maps. Both maps providers are implemented as base themes and to be able to use them, you must inherit their base theme in the theme you use for your site.

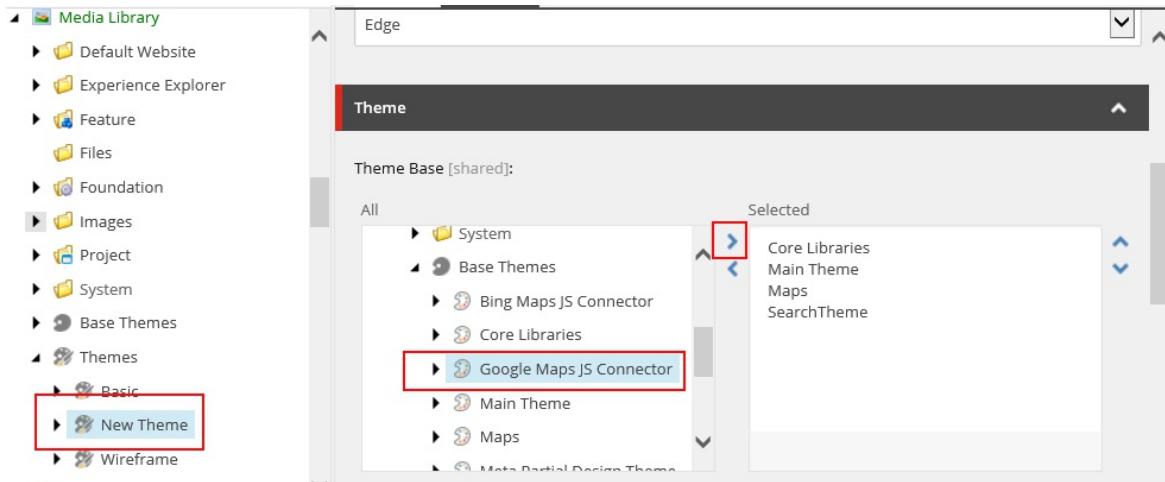
This topic describes how to:

- [Add the map base theme to a theme](#)
- [Store the maps authorization key](#)

[Add the map base theme to a theme](#)

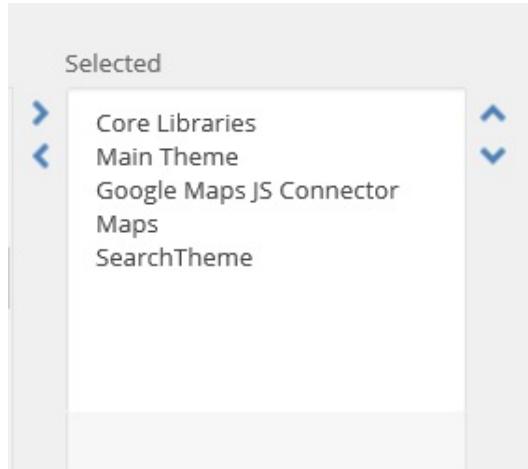
To add the map base theme to a theme:

- In the Media Library folder, in your custom theme, in the Theme section, select the maps provider from the Base Themes folder. You can only select one map provider per theme.



Note

The order of selected base themes is important. Be careful when you are adding something new or changing the order. The order of the themes must be: Core Libraries, Main Theme, Provider Theme, Maps, SearchTheme.



Store the maps authorization key

Google Maps API and Bing Maps API applications require authentication.

To store the maps authorization key:

- Go to Settings/Maps Provider, and in the Properties section, in the Key field, enter the authorization key.

The screenshot shows the Sitecore CMS interface. On the left, there is a navigation tree under the 'Settings' category. The 'Maps Provider' item is selected and highlighted with a blue background. Other items in the tree include 'Browser Title', 'Cookie Warning', 'Creative Exchange Storage Grouping', 'Datasource Configurations', 'Default links', 'Facets', 'Item Queries', and 'Redirects'. To the right of the tree, the 'Properties' pane is open, displaying basic item information:

Template:	/sitecore/templates/Foundation/E...
Created from:	[unknown]
Item owner:	[unknown]

The 'Properties' tab is active, and below it, there is a 'Key:' field containing a long, obscured API key.

Note

Depending on the maps provider you are using, refer to Google Maps or Bing Maps for more information about API keys.

Once the maps provider is configured correctly and added to the theme you use for your site, you can use the Map rendering on your pages.

Generate a custom static error page

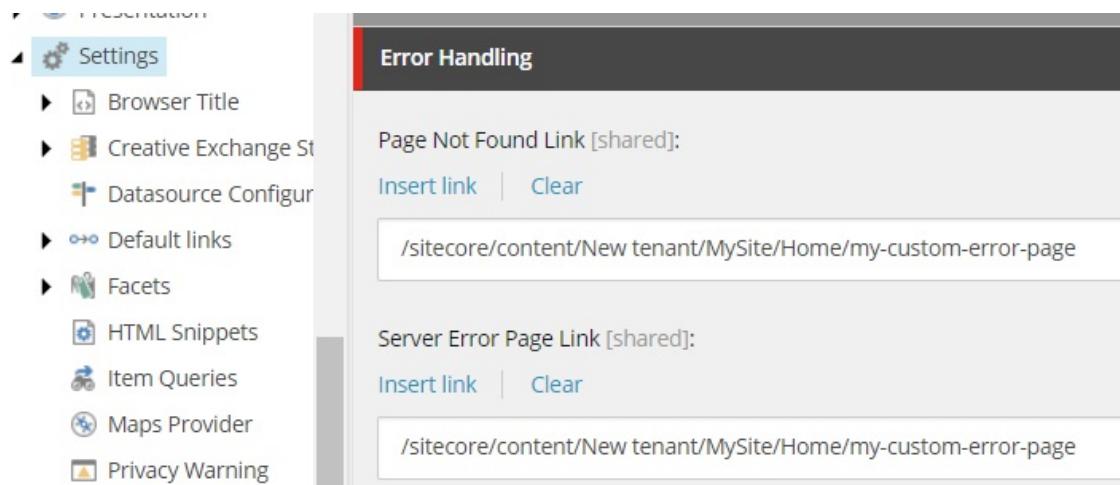
Abstract

Link to custom error pages.

SXA lets you set site-specific 404 and 500 pages. When an error occurs, visitors will be redirected to this static error page.

To generate a static:

1. Create your error page(s).
2. Publish the error page(s).
3. Navigate to `/sitecore/Content/Tenant/Site/Settings` and in the Error Handling section insert the links to your custom error pages.



Note

If you cannot find the Error Handling section, you may have forgotten to select the Error Handling option when creating your tenant.

4. Navigate to `/sitecore/Content/Tenant/Site/Settings/Site`

Grouping/Site and in the ribbon, on the Error Handling tab, click Generate static error page and click Ok. Now for every error that cannot be handled, visitors will see the static page that is loaded from the ErrorPages folder located in the root of your website.

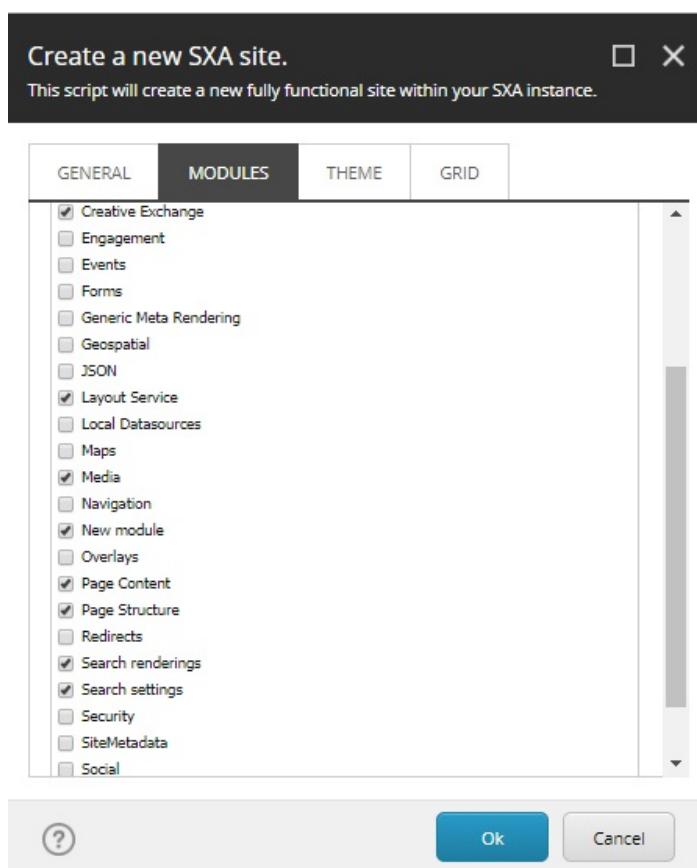
Restructure the SXA Toolbox

Abstract

Change the organization of components in the toolbox.

To make it easier to construct your page, SXA comes with a toolbox with reusable renderings that you can drag and drop onto your page. The SXA toolbox organizes all renderings in categories, such as Page content, Page structure, Navigation, Forms, and so on.

When creating a new site, you can determine which renderings will be available for that site.



SXA also lets you change the structure of the components within the toolbox.

This may be convenient when you want to move a component category that you use often to the top of the list or if you want to change the order of the components within the category.

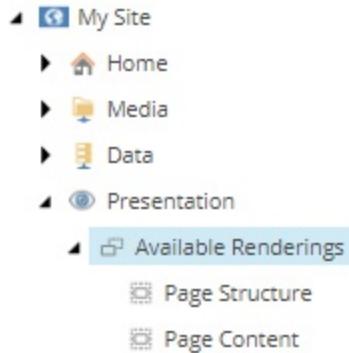
This topic describes how to:

- [Change the order of component categories](#)
- [Change the order of components within a category](#)
- [Add a custom category](#)

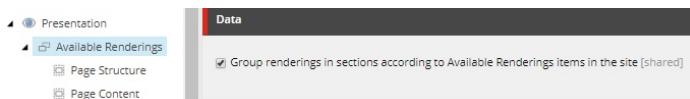
Change the order of rendering categories

To change the order of appearance of rendering categories in the Toolbox:

1. Navigate to sitecore/Content/Tenant/Site/Presentation, and in the Available Renderings item reorganize the rendering categories. For example, move the Page Content and Page Structure categories to the top of the list.



2. In the Data section, select the Group renderings in sections according to Available Renderings items in the site.



3. Save the changes. In the Experience Editor, the structure of the toolbox

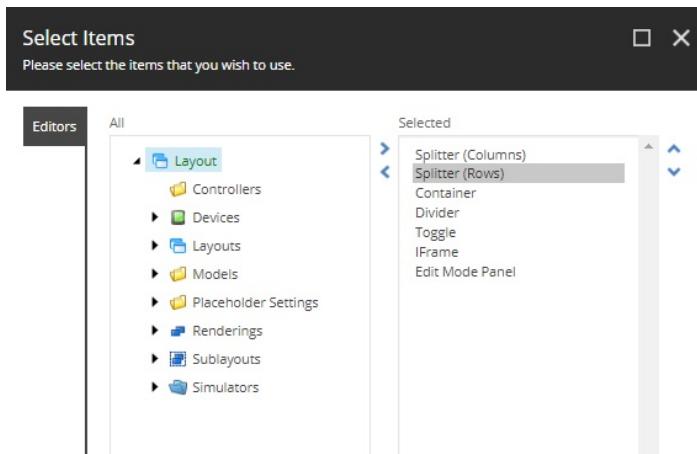
will be changed.



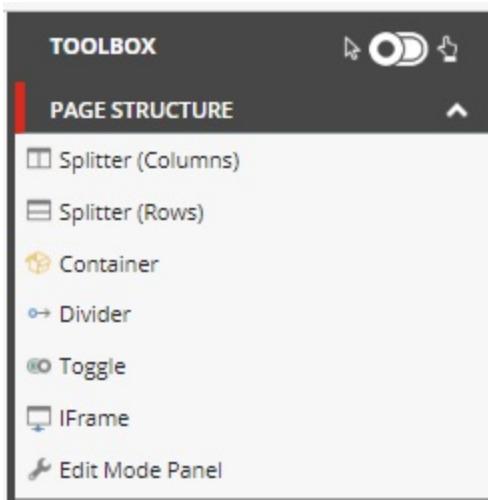
Change the order of renderings within a category

To change the order of the renderings within a category:

1. Navigate to `sitecore/Content/Tenant/Site/Presentation/Available Renderings` and click the category that you want to change the structure for.
2. In the Data section, in the Renderings field click Edit.
3. In the Select items dialog box, in the Selected field change the order of the components by using the arrows. For example, for the Page Structure category, move the splitter renderings to the top.



4. Save the changes. In the Experience Editor, the structure of the toolbox will be changed.



Add a custom category

To group renderings in a custom category:

1. Navigate to sitecore/Content/Tenant/Site/Presentation, right-click Available Renderings and click Available Renderings.
2. Enter a name and click Ok.
3. In the Data section, in the Renderings field click Edit.
4. In the Select items dialog box, navigate to Layout, Renderings and select the components that you want to add to your custom category.

Add a module to an existing tenant or site

Abstract

Install modules for existing tenants and/or sites using a script or the bulk module installer.

Add modules to tenants and sites using context menu script (**Add Tenant/Site Module**) or **bulk module installer**. When you create a new SXA tenant or site, the wizard lets you select the modules that you want to include. When you have an existing tenant or site and want to add modules later, you can add modules to a specific tenant or site or use the SXA bulk module installer to install modules for multiple tenants/sites.

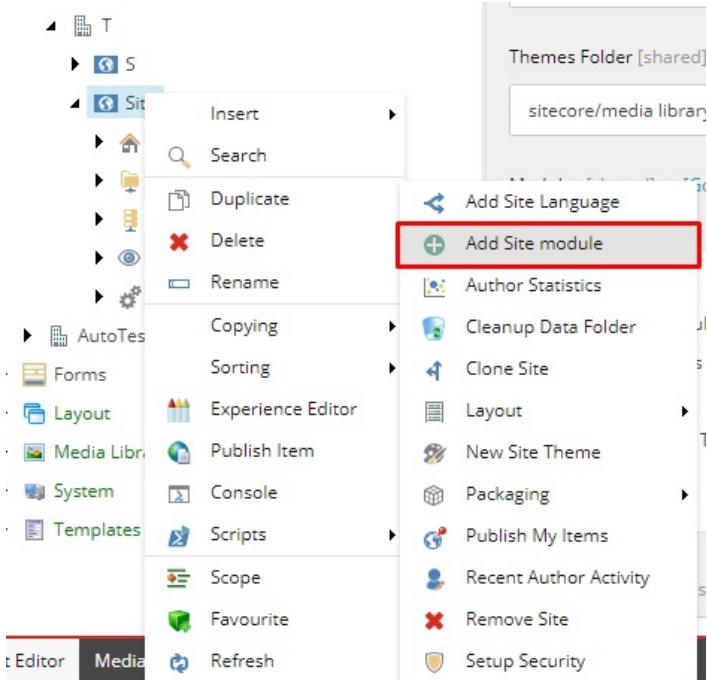
This topic describes how to:

- [Add modules to a specific tenant or site](#)
- [Add modules to multiple tenants and sites using the Bulk module installer](#)

[Add modules to a specific tenant or site](#)

To add modules to a specific tenant, site:

1. In the Content Editor, right-click the tenant or site, click Scripts and click Add tenant module/Add site module.

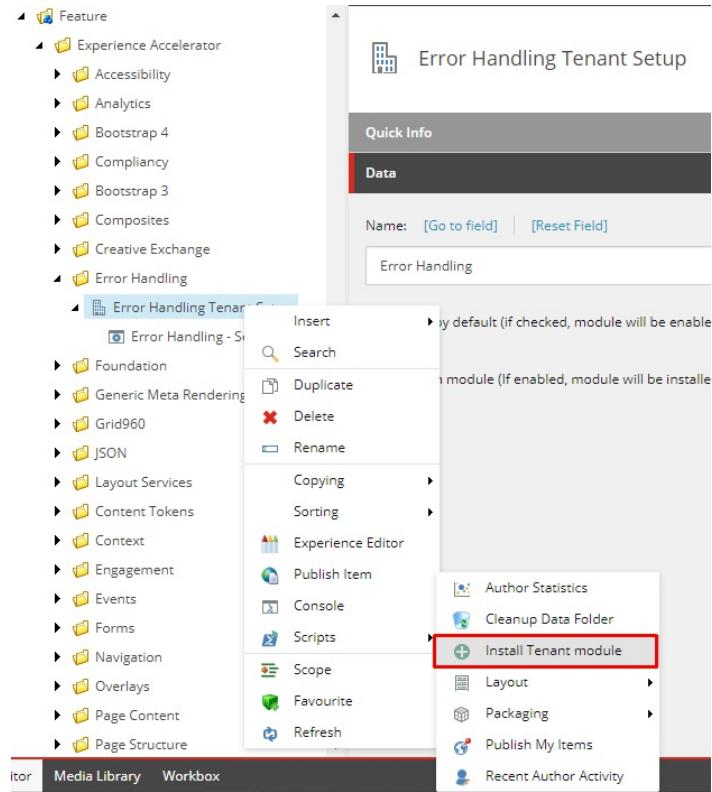


2. In the Add module to site/tenant dialog, the missing modules are listed. Select the modules that you want to add and click Ok.

Add modules to multiple tenants and sites using the Bulk module installer

To add a module to multiple tenants, sites:

1. In the Content Editor, navigate to the module that you want to add in /sitecore/System/Settings/Foundation/ or /sitecore/System/Settings/Feature/
2. Right-click the module, click Scripts, and click Install Tenant module/Install Site module.



3. In the Add module to site/tenant dialog select the tenants/sites that you want to add the module to and click Ok.

Exclude a page from the navigation menu

Abstract

Hide a page from navigation menu by using a navigation filter

You may want to hide some of your pages from the navigation menu. For example, when you created a static home page, or when you have form pages that visitors only need to see when they click a link. You can hide page using the custom navigation filters or create a new navigation filter. This topic describes how to:

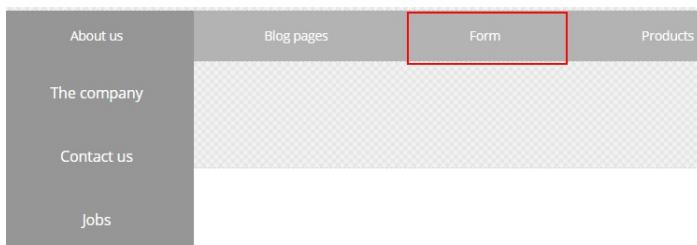
- [Hide a page using a navigation filter](#)
- [Create a custom navigation filter](#)

Hide a page using a navigation filter

SXA comes with navigation filters. You can use these existing filters to hide a page from the navigation menu.

To hide a page from the navigation menu:

1. For example, if the navigation rendering is used on a page that uses main navigation, it would normally show the Form page.



If you want to hide the form page, In the Content Editor, on the page level, you can select the navigation filter that you don't want the page to show up for. Navigate to the page and in the Navigation section, in the Check to hide in navigation filters field, select the filter you want to hide the page for.

The screenshot shows the Content Editor interface. On the left, there's a navigation tree under 'My site'. Items like 'Home', 'About us', 'Blog pages', and 'Form' are visible, with 'Form' being the selected item and highlighted by a red box. To the right, there's a 'Navigation' settings panel. It includes a 'Link Caption in Navigation:' field containing 'Form', a 'Check to hide in navigation filters [shared]' section with a checked checkbox next to 'Main Navigation' (also highlighted with a red box), and buttons for 'Select all', 'Deselect all', and 'Invert selection'.

2. In the Experience Editor, in the Navigation toolbar, click Edit component properties in the Navigation Filter field, select Main Navigation.

The screenshot shows the Experience Editor's 'Control Properties' dialog box. The 'Navigation Settings' tab is selected. It contains fields for 'Start Page' (empty), 'Top Navigation Level' (set to 1), 'Bottom Navigation Level' (set to 2), and a 'Navigation Filter' dropdown. The 'Main Navigation' option in the dropdown is highlighted with a blue selection bar. There are other options like 'Breadcrumb Navigation', 'Footer Navigation', 'Sidebar Navigation', 'Sitemap Navigation', and 'My custom navigation' in the list.

If you now reload the page, you will see that the Form page is removed from the navigation menu.

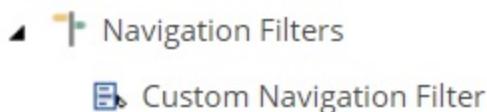


Create a custom navigation filter

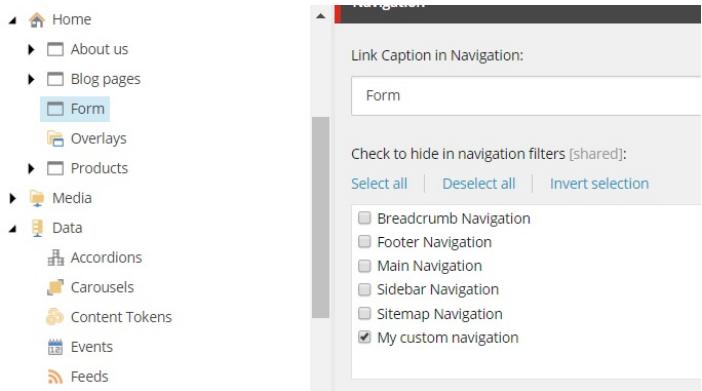
You can also create a custom navigation filter.

To create a custom navigation filter:

1. Navigate to /sitecore/content/Tenant/Site/Data, right-click Navigation Filters, and click Insert, click Navigation Filter.
2. Enter a name and click Ok.



3. At the page level, you can select the navigation filter that you don't want to page to show up for. In the Check to hide in navigation filters field, you can now select the filter you created and/or any other filter.



Adjust link settings to enable cross-site linking

Abstract

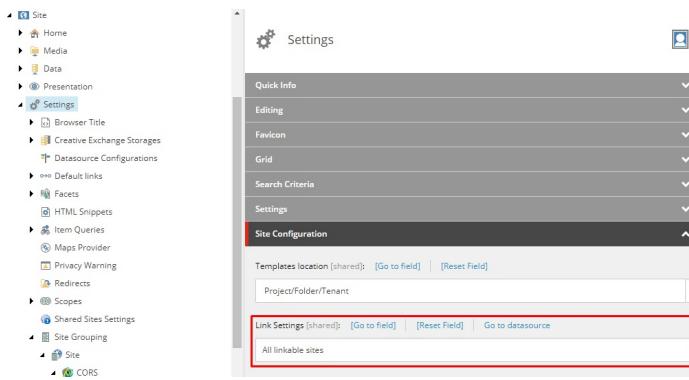
Enable linking between tenants and sites.

The Link rendering lets content authors add links to sites. Links can be internal links, media links, external link, anchors, mail, and JavaScript. To link within SXA tenants and sites, you must adjust the site settings.

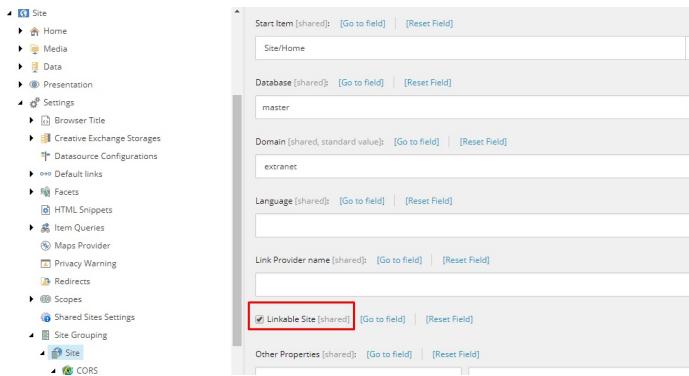
To enable cross-site linking:

1. Set the link root location by navigating to sitecore/Content/Tenant/Site/Settings.
2. In the Site Configuration section, in the Link Settings field, select one of the following options:

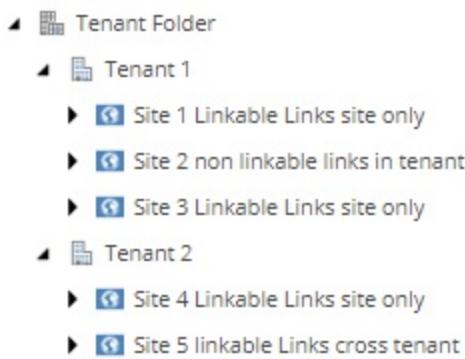
Link Settings field option	Link root location
Itself only	Default value. The available link root location comes from the current site.
Linkable sites in tenant	All linkable sites within the current tenant.
All linkable	All linkable site within all tenants.



3. To make a site linkable, navigate to `sitecore/Content/Tenant/Site/Settings/Site Grouping/Site`, and in the Basic section, select the Linkable Site check box.



For example, if you have five sites divided over two tenants:



Site 5 is set to be able to link across tenants.



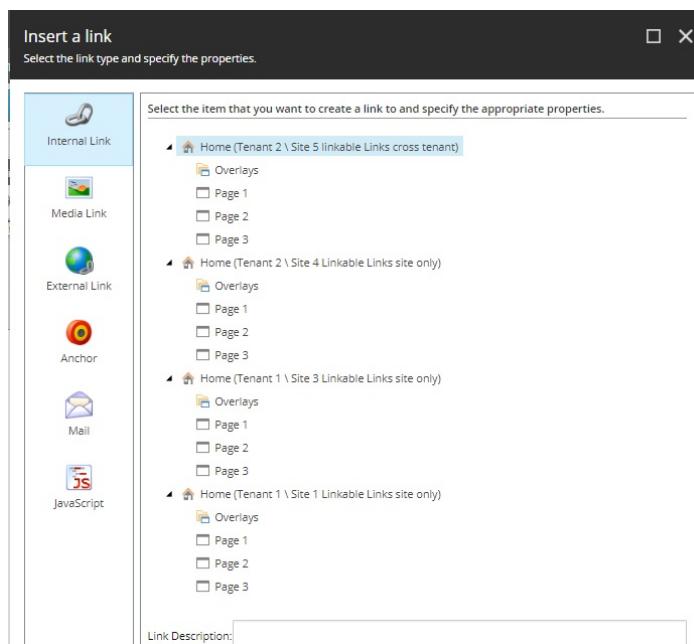
Project/Tenant 2

Link Settings [shared]:

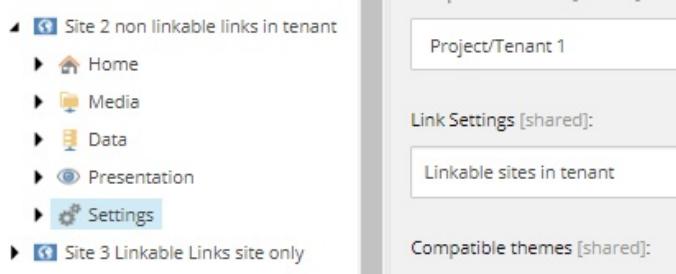
All linkable sites

All sites except site 2 are set to be linkable.

If you add the link component in the Experience Editor, and click Edit Link, in the Insert a link dialog box, you can link to all sites except for site 2.



Another example: Site 2 is set to not be linkable, and the Link Settings field is set to enable linking to sites within the same tenant:



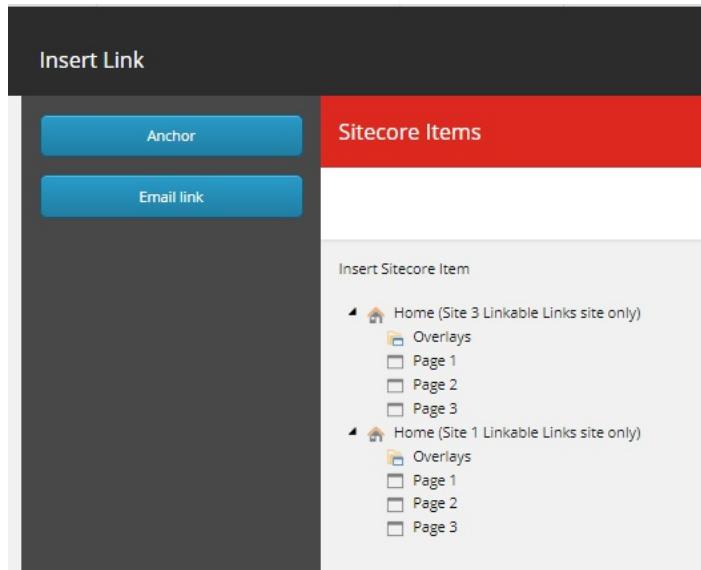
Project/Tenant 1

Link Settings [shared]:

Linkable sites in tenant

Compatible themes [shared]:

If you now insert a link in the Content Editor, you can only link to site 1 and site 3 within the same tenant.



Note

To make non-SXA sites linkable, you must add the `sxaLinkable="true"` property from SXA add the configuration node of the site. For example:

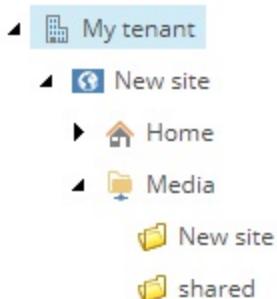
```
<site name="website" enableTracking="true" virtualFolder="/" phys
```

Configure the scope of the Media Library for an SXA site

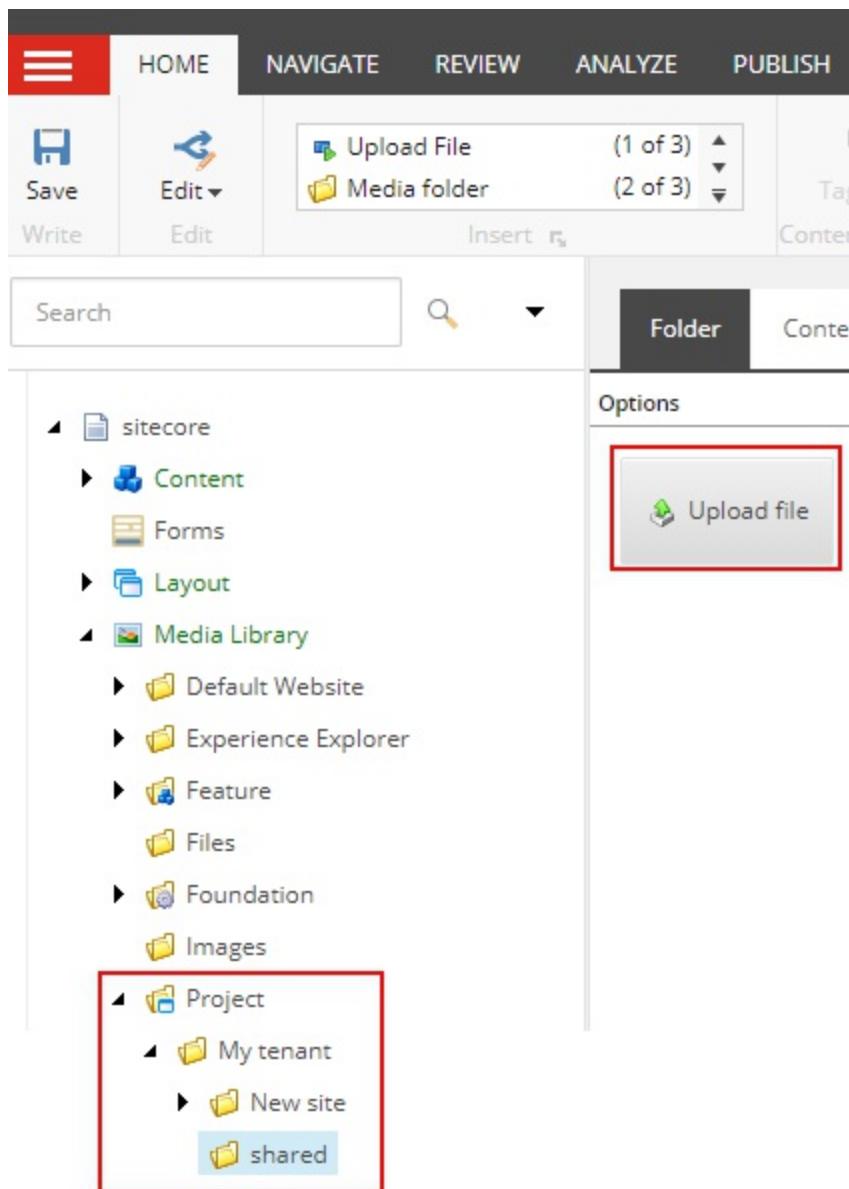
The *Media* folder in SXA sites (`sitecore/Content/Tenant/Site/Media`) is a scoped view of the Media Library folders available for the site. This view lets you determine which media items are available for your content editors for this particular site. This means that content editors can select media items that are relevant for the site rather than having to search through the complete Sitecore Media Library.

By default, when you add a site using the Site creation wizard, the following Media Library folders are added to the site's *Media* folder:

- The *virtual* folder on a site level (`sitecore/Content/Tenant/Site/Media`) by default contains two folders:
 - Site folder - contains the items available for this site only.
 - Shared folder - contains the items available for all sites within the tenant.



- The actual folder on the Media Library level: `sitecore/Media Library/Project/My tenant`. You can upload images here.



You can also make other folders from the Sitecore Media Library available to the content editors.

To add Media Library folders to your site *Media* folder:

- In the Content Editor, navigate to `sitecore/Content/Tenant/Site/Media` and in the Virtual section in the Additional Children field, move the relevant folder to the Selected field and click Save.

The image shows a SharePoint interface with a navigation menu on the left and a 'Virtual' ribbon tab on the right.

Navigation Menu:

- My tenant
- New site
 - Home
 - Media
 - New site
 - shared
 - Data
 - Presentation
 - Settings
- AutoTests

Ribbon Tab:

Virtual

Additional Children [shared]:

All Selected

- Media Library
- Default Website
- Experience Explorer
- Feature
- Files

Selected items: New site, shared, Default Website

Tenants

The SXA content architecture includes tenants and sites. SXA supports multitenancy, which means that you can run multiple sites on a single instance of Sitecore. Each tenant can include multiple related sites, for example, to support multiple brands for a single company or multiple languages or locations for a single brand.

Learn more about tenants with these topics:

- [Create a tenant and a site](#)
- [Add and extent an SXA template](#)
- [Export a tenant](#)
- [Add a module to an existing tenant](#)
- [Remove a tenant or a site](#)

Create a tenant and a site

Abstract

The SXA content architecture includes tenants and sites.

The SXA content architecture includes tenants and sites. SXA supports multitenancy, which means that you can run multiple sites on a single instance of Sitecore. Each tenant can include multiple related sites, for example, to support multiple brands for a single company or multiple languages or locations for a single brand. Organizations can support multiple languages through one-to-one translated versions (native Sitecore language support) or use a model with a separate site for each supported language.

For example, an international clothing company could have different tenants for the different brands of clothing and different sites for the specific countries.

This topic describes how to:

- [Create a tenant and a tenant folder](#)
- [Create a site and a site folder](#)

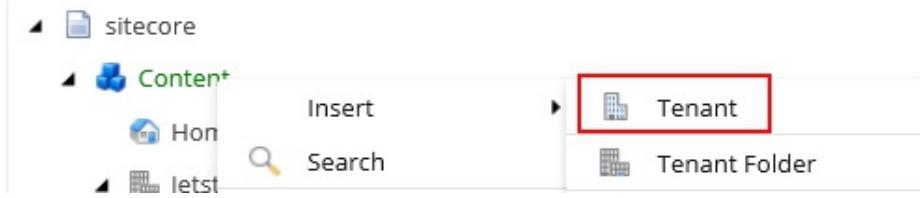
[Create a tenant and a tenant folder](#)

If you have multiple sites that need to share resources they must be created within one tenant. With SXA's multitenant architecture, you can provide each tenant a dedicated share of the Sitecore instance including its data templates, configuration, user management, tenant individual functionality, and non-functional properties.

To create a tenant:

1. In the Content Editor, right-click the content item in the content tree,

click Insert, Tenant.



2. In the wizard, enter a name for the tenant, select the features, and click OK.

For more complex solutions, you can use groups of tenants. For example, a multinational selling consumer goods could have the following tenant folders and tenants:

Company (Tenant Folder)

- Cosmetics (Tenant Folder)
 - Brand A (Tenant)
 - Brand B (Tenant)
- Laundry detergents (Tenant Folder)
 - Brand A (Tenant)
 - Brand B (Tenant)
 - Brand C (Tenant)
- Hair care (Tenant Folder)
 - Brand A (Tenant)
 - Brand B (Tenant)
 - Brand C (Tenant)
 - Brand D (Tenant)

To create a group of tenants:

- Right-click the content item in the content tree, click Tenant Folder, enter a name and click OK.

Note

If you want to move tenants between environments, you can package tenants with the [SXA Tenant Exporter](#).

Create a site and a site folder

The tenant is a top-level container for the sites underneath. Sites in the same tenant are related, for example, because they share the same set of templates or part of the media library. Sites are the items that represent the website and consist of pages, data, designs, and partial layouts.

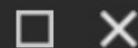
To create a site:

1. In the Content Editor, right-click the tenant to which you want to add the site.



2. In the wizard, on the General tab, enter the name for the site.
3. On the Features tab, select the features and click OK.
4. On the Theme tab, either create a new theme by selecting Create new theme or select one or more existing theme(s) and click Ok.

Create a new SXA site.



This script will create a new fully functional site within your SXA instance.

GENERAL

MODULES

THEME

GRID

Create a new theme.

New theme name

My Theme

The themes this site can use.

All

- ◀ Basic2
- ▶ gulp
- ▶ sass
- ▶ Scripts
- ▶ fonts
- ▶ images

Selected

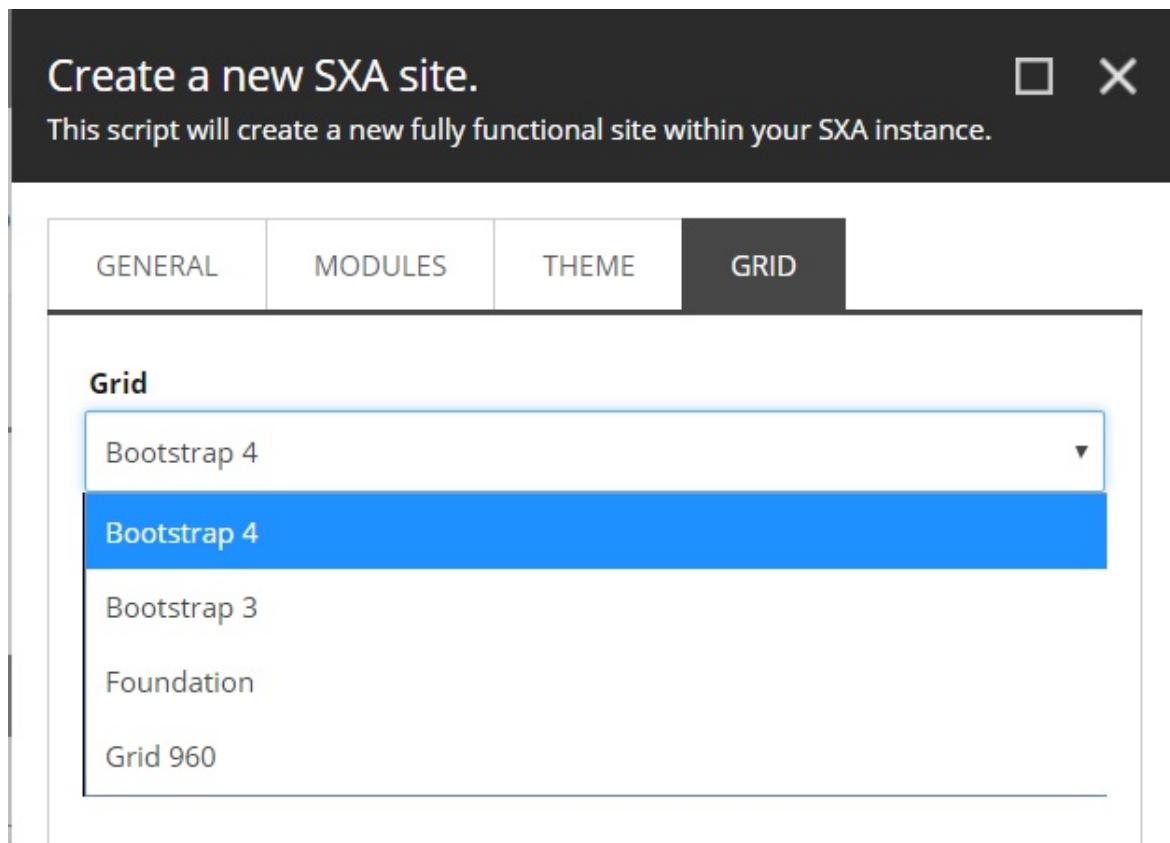
Wireframe



Ok

Cancel

5. On the Grid tab, select the grid and click OK.



Note

It is important to be aware that changing the grid system after you create your site requires many manual changes. Because of the references on your pages to the previous grid system, making a change to the grid system will break your layout.

Your new site is available immediately.

6. The SXA Site Manager dialog box displays a list of all SXA sites on your Sitecore instance. The icons indicate whether the site can be accessed or whether there is a conflict . If there are site conflicts that you need to solve, you can:

Move the site – click the site and on the ribbon, click Move Up. Use the icons move up and move down.

Correct the site definition – click the site and on the ribbon, click Open.

In the Basic section, change the site name, environment, virtual folder, or the host name.

Note

You can [create a duplicate](#) of a site, by using the clone script.

For governance reasons, you can decide to use groups of sites. For example, for an internalization model where you create different sites for different countries:

- Europe (site folder)
 - Poland (site)
 - Denmark (site)
 - The Netherlands (site)
 - Ukraine (site)
- Asia (site folder)
- Africa (site folder)

To create a group of sites:

- Right-click the tenant in the content tree, click Site Folder, enter a name and click OK.

Note

SXA defines sites as Sitecore items. This enables to mitigate an issue with website restart after each change in Sitecore.config file. You can create new sites with SXA without a need to restart your Sitecore instance. If for some reason you decide to define sites in the config file instead of using items, makes sure to add `IsSxaSite="true"` parameter to your site definition XML node. SXA will use this parameter to determine whether a defined site is an

SXA site or not. For example:

```
<site name="C" IsSxaSite="true" enableTracking="true" virtualFold
```

Add and extend an SXA template

Abstract

Create a new page template for your SXA solution.

Templates are preset schemas for content items that are used to base new items on instead of recreating the specific fields of a new item every time. You might want to add a data template when, for example, a project requires fields that are not defined in existing data templates, or when new items require unique default field values or default settings (for example, a default workflow).

Important

We recommend not to change the `_standard` values of predefined SXA templates. To avoid Experience Editor issues, it is better to create your own template and inherit from an SXA template.

Once you create a tenant, SXA will create a tenant templates location for you in a Project folder. You can add new templates for specific projects in that same folder. This may be convenient if you are working on a project that needs custom templates. You can add your project templates to:
`/sitecore/templates/Project/`

Note

We strongly recommend that you adhere to the Helix guidelines. Helix offers a set of official guidelines and recommended practices for Sitecore development.

SXA validates items by checking an item's template inheritance instead of template name or template ID. This makes it possible for you to create and/or extend a template without breaking anything.

This topic describes how to:

- [Create an SXA template](#)
- [Extend an SXA template](#)

Create an SXA template

To add a template:

1. In the Content Editor, navigate to your tenant templates folder (sitecore/Templates/Project/). For example, to add a custom template to your new project, go to: /sitecore/templates/Project/projectname.

Important

Never put custom templates in SXA-controlled branches because SXA overwrites these during SXA updates. For example, you can place templates in folders that are siblings of the Experience Accelerator folder but not inside.

2. Right-click the item and click Insert, New Template. Alternatively, click NewTemplate in the Options section.
3. Enter a name for the new template, select a base template, and click Next. For example, templates for items that represent a page need to inherit from the default Page template.

Note

Although it is not required to follow SXA naming conventions, you might find it useful as it clearly defines how a particular template can be used. In SXA, template names that cannot be used to create items but can function as base templates for other templates are prefixed with an underscore. For example: _Searchable, _Designable, _Sticky Note.

4. To add additional base templates to your template when they define

standard fields that you want your new template to inherit, click the template and on the Content tab, in the Data section, double-click the template you want to add, or use the arrow to add it.

5. On the Builder tab, in the Add a new section field, add the relevant data template fields, for example, Data.

Note

To assign default values to fields in your template, on the Options tab, click Standard values. With each new item created from that template, fields will inherit values from the corresponding field in the standard values item.

6. Save your changes.

Extend an SXA template

You can extend an existing SXA template. Templates can consist of many base templates.

Note

You must make sure that you do not break the template inheritance. You can see the inheritance structure of the template in the Content Editor, on the Inheritance tab of the template.

To extend an existing SXA Project template with your custom template:

1. In the Content Editor, click the template that you want to extend.
2. On the Content tab, in the Data section, double-click the template you want to add and move it to the Selected section using the arrow.

Note

If the SXA template that you want to extend does not exist in your Project

templates folder yet, it is better to create a new Project template and use the Base template field to add the template that you want to extend as well as the custom base template that you added.

Export a tenant

Abstract

Create a package that contains the tenant content, templates, themes, and media.

Packaging tenants with the SXA Tenant Exporter is the easiest way to move tenants between environments. With the Tenant Exporter, you can quickly create a package that contains:

- The tenant content – /sitecore/content/Group/Tenant
- The tenant templates – /sitecore/templates/Project/Group/Tenant
- The tenant themes – /sitecore/media library/Themes/Group/Tenant
- The tenant media library items – /sitecore/media library/Project/Group/Tenant

To export a tenant:

1. In the Content Editor, right-click the tenant item, click Scripts, and click Quick Download Tenant as Package.
2. In the Download Tenant as package dialog box, enter the metadata and click OK.
3. When the export has finished, click Download.

Add a module to an existing tenant or site

Abstract

Install modules for existing tenants and/or sites using a script or the bulk module installer.

Add modules to tenants and sites using context menu script (**Add Tenant/Site Module**) or **bulk module installer**. When you create a new SXA tenant or site, the wizard lets you select the modules that you want to include. When you have an existing tenant or site and want to add modules later, you can add modules to a specific tenant or site or use the SXA bulk module installer to install modules for multiple tenants/sites.

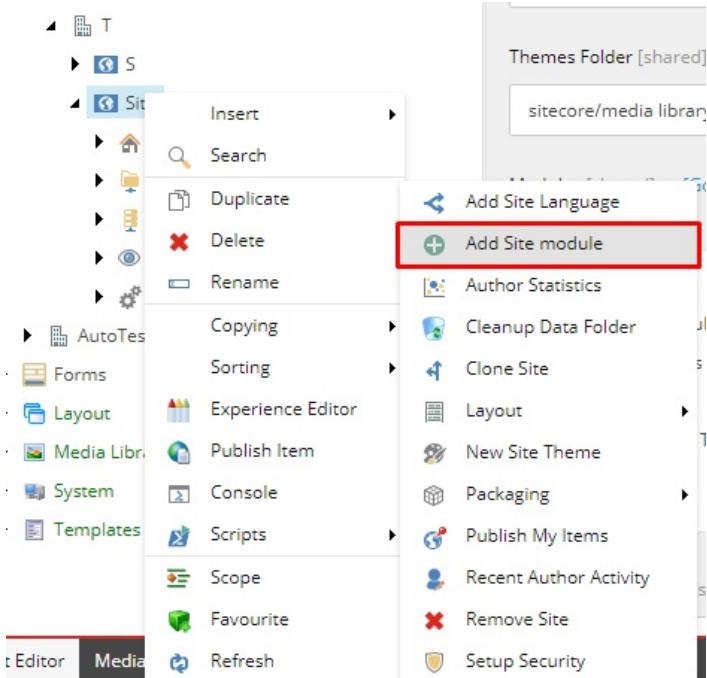
This topic describes how to:

- [Add modules to a specific tenant or site](#)
- [Add modules to multiple tenants and sites using the Bulk module installer](#)

[Add modules to a specific tenant or site](#)

To add modules to a specific tenant, site:

1. In the Content Editor, right-click the tenant or site, click Scripts and click Add tenant module/Add site module.

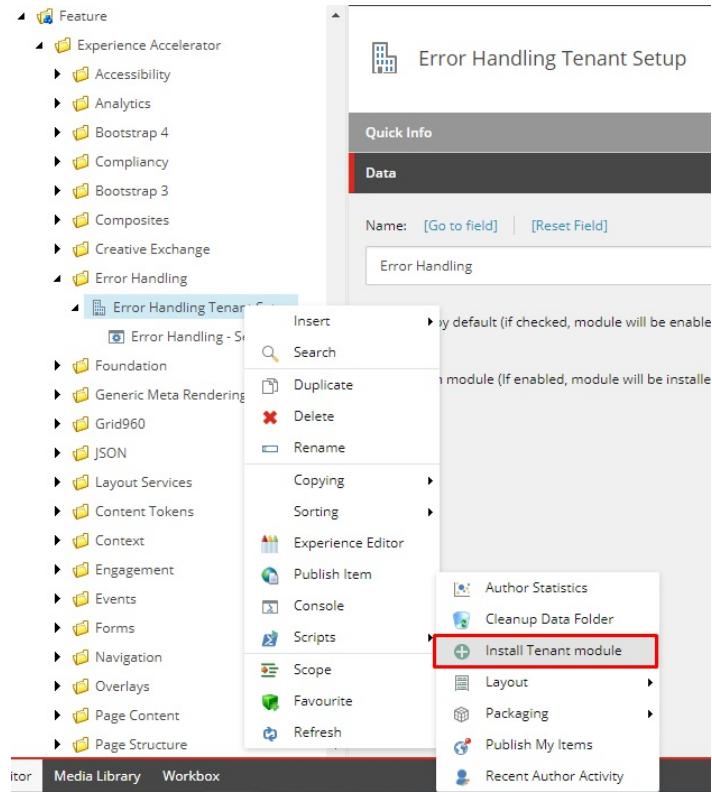


2. In the Add module to site/tenant dialog, the missing modules are listed. Select the modules that you want to add and click Ok.

Add modules to multiple tenants and sites using the Bulk module installer

To add a module to multiple tenants, sites:

1. In the Content Editor, navigate to the module that you want to add in /sitecore/System/Settings/Foundation/ or /sitecore/System/Settings/Feature/
2. Right-click the module, click Scripts, and click Install Tenant module/Install Site module.



3. In the Add module to site/tenant dialog select the tenants/sites that you want to add the module to and click Ok.

Remove a tenant or a site

Abstract

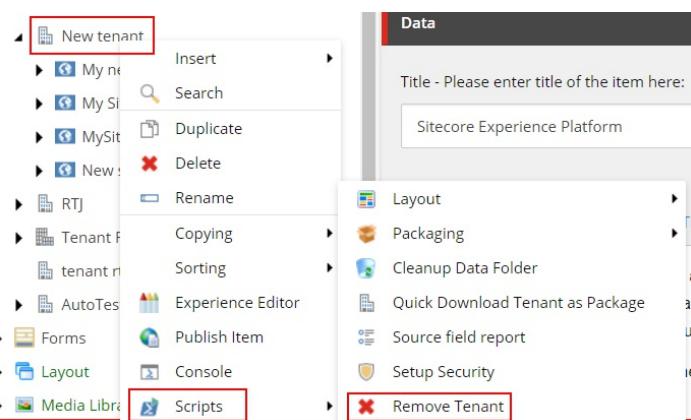
In SXA, you can remove tenants and sites and all related items by using scripts.

Inevitably, you will need to remove tenants and/or sites from time to time. In SXA, you can use a script to remove tenants (folders) and sites (folders) including all the media items and template folders that are created during the scaffolding process. Depending on what you want to remove, there are scripts available to remove tenant folders, tenants, site folders, and sites.

To remove a tenant, site, or folder:

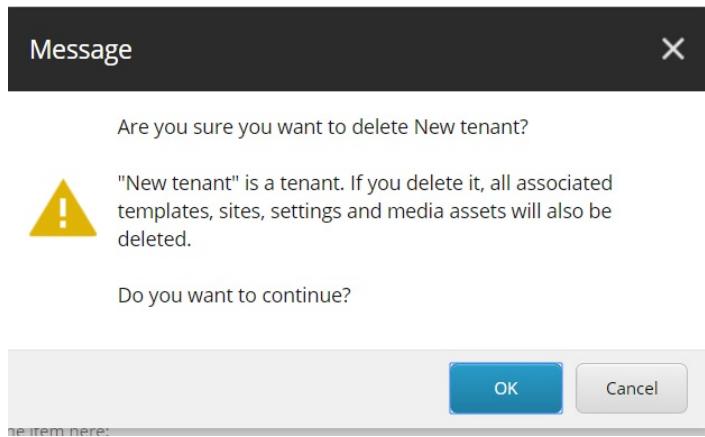
1. In the Content Editor, right-click the tenant, site, or folder that you want to remove.
2. Click Scripts and then, depending on what you want to remove, click Remove Tenant, Remove Site, Remove Tenant Folder, or Remove Site Folder.

For example, to remove the *New tenant* tenant:



3. In the Message dialog box, to confirm that you want to remove, for

example, the tenant, click OK.



SXA now runs a script to remove the tenant, site, or folder and all the related templates and media files.

Extending SXA

You might want to extend SXA, for example, to add your own custom rendering to the SXA toolbox, add a theme, or a template.

You can extend the logic of SXA processing pipelines and message events with your own implementation. You can extend and customize the pipelines by adding or replacing processors.

Note

If you want to change or extend any of the SXA functionality, you must patch it in a correct way, using Sitecore patch configuration files.

Learn more about extending SXA by reading the following topics:

- [Create a custom grid](#)
- [Change or add database content](#)
- [Add a dynamic placeholder to a rendering](#)
- [Add modules to site and tenant scaffolding](#)
- [Build a rendering that includes variants](#)
- [Copy and customize a rendering](#)
- [Data sources](#)
- [The SXA pipelines](#)
- [The SXA script library](#)
- [Walkthrough: Building a simple rendering](#)

- [Create a custom grid](#)

Development

Learn more about extending SXA by reading the following topics:

- [Create a custom grid](#)
- [Change or add database content](#)
- [Add a dynamic placeholder to a rendering](#)
- [Add modules to site and tenant scaffolding](#)
- [Build a rendering that includes variants](#)
- [Copy and customize a rendering](#)
- [Data sources](#)
- [The SXA pipelines](#)
- [The SXA script library](#)
- [Walkthrough: Building a simple rendering](#)
- [Customize rendering HTML per site](#)
- [Embed HTML in SXA](#)
- [Create a custom token for a rendering variant](#)
- [Add a data source setting for an existing rendering](#)

Add a dynamic placeholder to a rendering variant

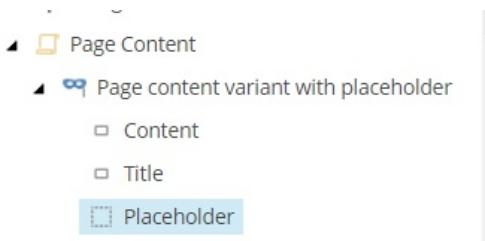
Abstract

Use dynamic placeholders to add the same placeholder name several times.

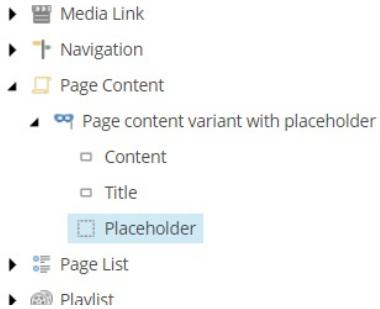
SXA lets you create rendering variants that render a placeholder that enables users to add additional renderings. You can add restrictions to the placeholder settings. For example, because you only want to content author to add an image rendering. When you use dynamic placeholders, you can add the same placeholder name several times across multiple renderings or within a single rendering. Dynamic placeholders guarantee that unique keys are used across different renderings and within one rendering.

To add a dynamic placeholder to a rendering variant:

1. Add a rendering variant. For example, add a variant for the Page Content rendering.
2. Right-click the variant, click Insert and click Placeholder.



3. In the VariantDetails section, add the Placeholder Key.



VariantDetails

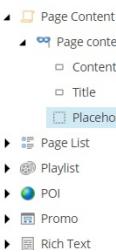
Css Class [shared]:

Placeholder Key [shared]:

Note

Do not use a dash in your Placeholder Key name. Using dashes followed by a number disables the rendering.

4. If you want the user to be able to switch context for the renderings inside this placeholder, go to the VariantDetails section and select the Switch component context to currently rendered item checkbox.



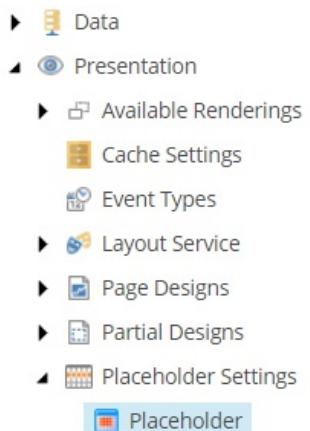
VariantDetails

Css Class [shared]:

Placeholder Key [shared]:

Switch component context to currently rendered item [shared]

5. To reference the placeholder key that you just created in the Placeholder Settings, go to Presentation/Placeholder Settings and add a placeholder.
6. In the Data section enter the Placeholder Key that you just created and add an * to make it dynamic and guarantee a uniqueness to the placeholder key across different renderings.
7. If you want to restrict the use of renderings in this placeholder, in the Data section, in the Allowed Controls field, click Edit and select the renderings that are allowed. For example, go to Layout/Renderings/Feature/Experience Accelerator/Media and select the Image rendering.



A screenshot of the Placeholder Settings edit screen. The title bar says "Data". Below it, there is a placeholder key input field with "myplaceholder*" and an "Edit" button. Under "Allowed Controls [shared]", there is a single item labeled "Image" with a small icon.

8. If you now add the Page Content rendering to the page and select the variant that you just created, you will see that the only available rendering that you are allowed to add is the Image:

A screenshot of the "Select a Rendering" dialog. The title bar says "Select a Rendering" and the sub-instruction "Select the rendering that you want to use. Click Select to continue." Below this, there is a "MEDIA" tab and a list of renderings. One rendering is visible: "Image" with a thumbnail icon showing a landscape scene.

Change or add database content

SXA follows the Sitecore Helix principles that contain development process recommendations for building, testing, extending, and maintaining Sitecore implementations. One of the Helix principles is that each module has a separate location in the content tree:

`{PATH_TO_SITECORE_LOCATION}/{LAYER}/{PRODUCT}/{MODULE}`

For example:

/sitecore/templates/Foundation/Experience Accelerator/Multisite

Where:

Parameter	Example value	Description
PATH_TO_SITECORE_LOCATION	/sitecore/templates/	Path to the location where project-specific items will be stored
LAYER	Foundation	Helix recognizes several different layers: <i>Project</i> , <i>Feature</i> , <i>Foundation</i>
		Optional. Additional level

PRODUCT	Experience Accelerator	of nesting for better organization of all SXA modules under a single folder.
MODULE	Multisite	Module name.

You can add new templates for specific projects. This can be convenient if you are working on a project that needs custom templates. You can add your project templates to: /sitecore/templates/Project/Project_Name

Add modules to Site and Tenant scaffolding

Abstract

Extend SXA by adding scaffolded items.

Scaffolding lets you add modules to [sites and tenants](#). SXA modules are stored in the Feature or Foundation folder:

- /sitecore/System/Settings/Foundation/Experience Accelerator/
- /sitecore/System/Settings/Feature/Experience Accelerator/

To define your new module, you must add the Site Setup, Tenant Setup, or Grid Setup item to the module and add action types from the Scaffolding folder (`sitecore/Templates/Foundation/Experience Accelerator/Scaffolding/`).

This topic describes how to:

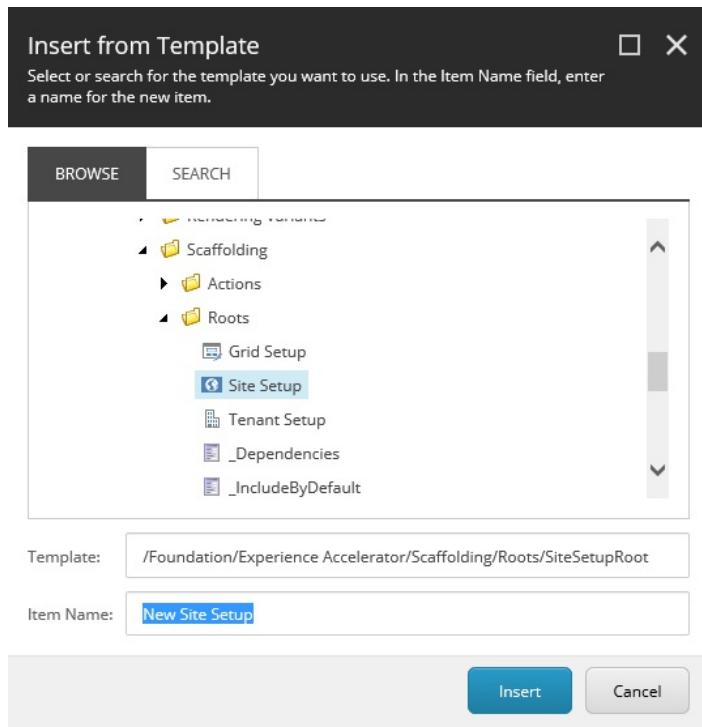
- [Add a module scaffolding definition](#)
- [Add scaffolding actions](#)

[**Add a module scaffolding definition**](#)

To add your own module scaffolding definition:

1. Navigate to /sitecore/System/Settings/Foundation/ or /sitecore/System/Settings/Feature/ and add a folder.
2. Right-click the new folder, and click Insert from template.
3. In the Insert from template dialog box, navigate to

sitecore/Templates/Foundation/Experience Accelerator/Scaffolding/Roots and, depending on the type of module that you want to add, click Grid Setup, Site Setup, or Tenant Setup, and then click Insert.



4. In the Data section, depending on the type of module that you want to install, fill in the following fields:

Setup	Field	Descriptions
-------	-------	--------------

Name		The name of the site module as you want it to display in the site creation wizard.
------	--	--

Dependencies		Specify the order in which the modules are installed.
--------------	--	---

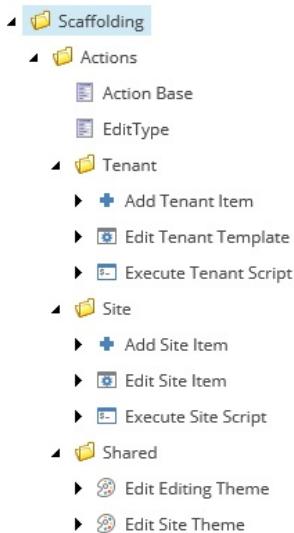
	Include by default	Select to enable the module by default in the site creation wizard.
Site Setup	Include if installed	Depending on whether the selected action was applied to the current tenant, the module will be automatically included. If the Selected field is empty, you can decide whether to install this module in the new-site creation wizard.
	Is system module	Select to install the module automatically. In this case, the module does not appear in the dialog box.
	Name	The name of the tenant module as you want it to display in the tenant creation wizard.
Tenant Setup	Include by default	Select to enable the module by default in the tenant creation wizard.
	Is system module	Select to install the module automatically. In this case, the module does not appear in the tenant creation wizard.
	Name	The name of the grid system as you want it to display in the site creation wizard.
Grid Setup	Dependencies	Specify the order in which the modules are installed.

Grid Definition

Refers to the Grid Definition item. For example, for the Foundation grid system:
Settings/Feature/Experience
Accelerator/Foundation/Foundation

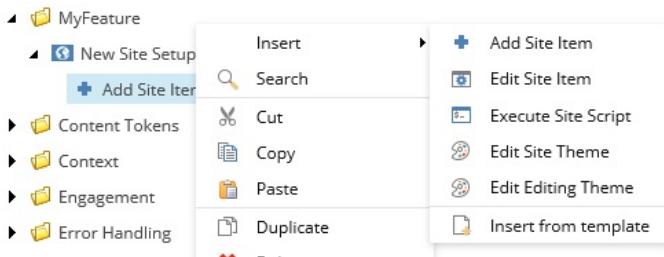
Add scaffolding actions

You can use various action types to define modules. You can add items, edit the template, and execute PowerShell scripts. The available actions are stored in the Actions folder (sitecore/Templates/Foundation/Experience Accelerator/Scaffolding/Actions):



To add a scaffolding action:

- Navigate to your new module and right-click the setup item that you added for your new module. For example, right-click the Site Setup item and insert the Add Site Item action.



The following actions are available:

Action type	Field	Description
Add Tenant Item	Location	Adds the new item under the tenant.
Add Tenant Item	Template	Template used to create the new tenant item.
Add Tenant Item	Name	Name of the item.
Add Tenant Item	Fields	Field/value mapping to set the fields of the new item after creation.
Add Tenant Item	Template	Select the template to copy into tenant templates. The template must be defined settings.
Edit Tenant Template	Type of action	Select the action type.
Edit Tenant Template	Base templates or insert	Select items to use as an argument for the action selected options

templates

Select the PowerShell script to execute. You can use scripts to automate the site/tenant creation process. For example:

Execute
Tenant
Script

Script

```
function Invoke-ModuleScriptBody {  
    [CmdletBinding()]  
    param(  
        # Depending on context could be a Tenant  
        [Parameter(Mandatory=$true, Position=0 )]  
        [Item]$Root,  
        [Parameter(Mandatory=$true, Position=1 )]  
        [Item[]]$TenantTemplates  
    )  
    begin {  
        Write-Verbose "Cmdlet Invoke-ModuleScriptBody -"  
    }  
    process {  
        Write-Verbose "Cmdlet Invoke-ModuleScriptBody -"  
        # Script body  
        # Put your custom logic here  
    }  
    end {  
        Write-Verbose "Cmdlet Invoke-ModuleScriptBody -"  
    }  
}
```

Location Adds the new item under the site.

Add Site
Item

Template Template used to create a new site item.

Name Name of the new site item.

Fields Field/value mapping to set the fields of the new item after creation.

Edit Site

Template Edits Site items by adding additional insert options.

Item

Type of action	Select the action type.	
Insert options	Define the insert options for the item.	
Execute Site Script	Select the PowerShell script to execute. You can use <code>sca</code> in the site/tenant creation process. For example:	
	<pre>function Invoke-ModuleScriptBody { [CmdletBinding()] param(# Depending on context could be a Tenant [Parameter(Mandatory=\$true, Position=0)] [Item]\$Root, [Parameter(Mandatory=\$true, Position=1)] [Item[]]\$TenantTemplates) begin { Write-Verbose "Cmdlet Invoke-ModuleScriptBody -" } process { Write-Verbose "Cmdlet Invoke-ModuleScriptBody -" # Script body # Put your custom logic here } end { Write-Verbose "Cmdlet Invoke-ModuleScriptBody -" } }</pre>	
Edit Editing Theme	Base Themes	List of base themes to add to an editing theme instance.

Edit Site Base List of base themes to add to an editing theme instance.
Theme Themes

Post Setup Script Step

Select the PowerShell script to execute. You can use `sca` once the site is created.

```
function Invoke-Step {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, Position = 1)]
        [Sitecore.XA.Foundation.Scaffolding.Mod
    )
    begin {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
    process {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
    end {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
}
```

Input Validation Script Step

Select the PowerShell script to execute. You can use `sca` before the site is created. If the result returned by any of Site dialog box will display again so that you can correct values.

```
function Invoke-Validation {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, Position = 1)]
        [Sitecore.XA.Foundation.Scaffolding.Mod
    )
    begin {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
    process {
        Write-Verbose "Cmdlet Invoke-Validation"
        # Return $true or $false as a result of
        $true
    }
}
```

```
        end {
            Write-Verbose "Cmdlet Invoke-Validation
        }
```

Select the PowerShell script to execute. You can use `sca` once tenant is created.

Post Setup Script Step

```
function Invoke-Step {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, Position
        [Sitecore.XA.Foundation.Scaffolding.Mod
    )
    begin {
        Write-Verbose "Cmdlet Invoke-Validation
    }
    process {
        Write-Verbose "Cmdlet Invoke-Validation
    }
    end {
        Write-Verbose "Cmdlet Invoke-Validation
    }
}
```

Select the PowerShell script to execute. You can use `sca` before the tenant is created. If the result returned by any New Tenant dialog box will display again so that you can selected values.

Input Validation Script Step

```
function Invoke-Validation {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, Position
        [Sitecore.XA.Foundation.Scaffolding.Mod
    )
    begin {
        Write-Verbose "Cmdlet Invoke-Validation
    }
    process {
        Write-Verbose "Cmdlet Invoke-Validation
        # Return $true or $false as a result of
        $true
    }
}
```

```
        end {
            Write-Verbose "Cmdlet Invoke-Validation
        }
    }
```

Select the PowerShell script to execute. You can use `sca` the site/tenant removal process. This action will be executed if the Remove-Site cmdlet are invoked. You can use it to clear modules because they are not removed by default removing the items are in different folders)

Pre Delete Step Script

```
function Invoke-Step {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true, Position = 1)]
        [Item]$Root
    )
    begin {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
    process {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
    end {
        Write-Verbose "Cmdlet Invoke-Validation"
    }
}
```

Build a rendering that includes variants

Abstract

Learn about the classes that are required for a new rendering that includes variants.

To create a new rendering that includes variants, you create a custom rendering, but use different base classes for controllers and repositories.

In the Presentation folder of your site, you can set which renderings are visible in the Toolbox in the Experience Editor. To be able to use the new custom rendering with its rendering variants, you must create the item using the exact same name under both the Available Renderings and the RenderingVariants folder.

This topic describes the changes needed in the code for renderings that include variants for the following classes:

- [Controller](#)
- [Repository](#)
- [Model](#)
- [View](#)

[Controller](#)

After you create a new rendering using SXA's basic structure, you must create a controller. For a rendering that includes variants, instead of inheriting from the StandardController class, the controller must inherit from the VariantsController.

For example, for a custom rendering called PageTitle the controller could look like this:

```
using PageTitle.Repositories;
using Sitecore.XA.Foundation.RenderingVariants.Controllers;
namespace PageTitle.Controllers
{
    public class PageTitleController : VariantsController
    {
        protected readonly IPpageTitleRepository pageTitleRepository;
        public PageTitleController(IPpageTitleRepository pageTitle)
        {
            pageTitleRepository = pageTitle;
        }
        protected override object GetModel()
        {
            return pageTitleRepository.GetModel();
        }
    }
}
```

Repository

The VariantsController provides the VariantsRepository:

```
using PageTitle.Models;
using Sitecore.XA.Foundation.Mvc.Repositories.Base;
using Sitecore.XA.Foundation.RenderingVariants.Repositories;
namespace PageTitle.Repositories
{
    public class pageTitleRepository: VariantsRepository, IPpageTitle
    {
        public override IRenderingModelBase GetModel()
        {
            pageTitleModel model = new pageTitleModel();
            FillBaseProperties(model);
            model.CustomProperty = "This is just an example rende
            return model;
        }
    }
}
```

Model

The model class of the new rendering must inherit from the

VariantsRenderingModel:

```
using Sitecore.XA.Foundation.Variants.Abstractions.Models;
namespace PageTitle.Models
{
    public class PageTitleModel: VariantsRenderingModel
    {
        public string CustomProperty { get; set; }
    }
}
```

View

To generate the HTML markup in your view rendering cshtml file, iterate through the item properties of the model.

For example:

```
@using Sitecore.Extensions
@using Sitecore.XA.Foundation.MarkupDecorator.Extensions
@using Sitecore.XA.Foundation.SitecoreExtensions.Extensions
@using Sitecore.XA.Foundation.RenderingVariants.Extensions
@using Sitecore.XA.Foundation.RenderingVariants.Fields
@using Sitecore.XA.Foundation.Variants.Abstractions.Fields
@model PageTitle.Models.PageTitleModel
<div @Html.Sxa().Component("page-title", Model.Attributes)>
    <div class="component-content">
        <h1>@Model.CustomProperty</h1>
        @foreach (BaseVariantField variantField in Model.VariantF
        {
            @Html.RenderingVariants().RenderVariant(variantField,
        }
    </div>
</div>
```

Copy and customize a rendering

Abstract

Learn how scaffolding lets you add custom renderings.

With SXA, you can quickly create a copy of an SXA rendering by using the clone script. With a cloned rendering, you have an exact copy of the rendering definition item, rendering parameters template, data source templates, and branches. You copy the existing rendering, and you name and style it differently. This can be convenient, for example, when you want many Promo renderings that are all styled differently.

Because the standard SXA toolbox sections are overwritten with SXA updates, we recommend that you create a separate toolbox section for derivative renderings. You can do this by adding a new module to SXA.

This topic describes how to:

- [Add a new module](#)
- [Copy and customize a rendering](#)
- [Add a rendering to the toolbox](#)

[Add a new module](#)

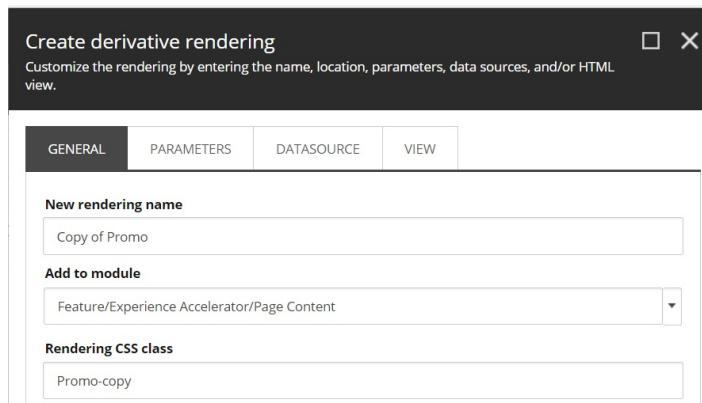
To add a new module:

1. Navigate to /sitecore/system/Settings, right-click Features or Foundation, and insert a new folder.
2. Right-click the new folder, click Insert, and click Module.
3. Enter a name and group the module under the new folder.

Copy and customize a rendering

To copy and customize a rendering:

1. To clone the rendering that you want to use, navigate to /sitecore/Layout/Renderings/ and right-click the rendering that you want to clone.
2. Click Scripts, and click Clone Rendering.
3. In the Create derivative rendering dialog box, fill in the following fields:



	Tab	Field	Description
	New rendering name	Enter the name for the new rendering.	
General	Add to module	Select the module the new rendering belongs to. Because the standard SXA toolbox sections are overwritten with SXA updates, we recommend that you add your own module for derivative renderings.	
	Rendering		

Parameters	Rendering Parameters
Datasource	Datasource

CSS class Enter the CSS class for the rendering.

Select the rendering parameters copy mode:

- Inherit from existing rendering parameters— creates a new item and sets the __Base template to point to the source rendering parameters template.
- Make a copy of original rendering parameters— creates a copy of the source rendering parameters in a new location. This enables you to extend, remove, and/or reorder parameters.

Select the data source copy mode:

- Inherit from existing datasource – creates a new item and set its __Base template to point to the source data source template. Select this option when you want to change the styling of the rendering only.
- Make a copy of original datasource – creates a copy of the source data source template in a new location. Select this option when you want to extend the standard data item with additional fields. For example, if you want to use an image field as a background.

Select the view copy mode:

- Use original MVC view file – uses the view from the prepopulated Path to the rendering view field. If the View field is not specified then

the StandardController tries to resolve the view based on rendering the item name. Select this option when you only want to change the styling.

View

- Copy MVC view file (specify path below) – copies the source view to a location specified in the Path to rendering view field. Select this option when you want to extend and/or modify the existing markup.
- Select existing MVC view file (specify path below) – uses the view specified in the Path to rendering view field but does not create a copy.

View

Enter the location of the view. For example:

Path to rendering view

GENERAL	PARAMETERS	DATASOURCE	VIEW
View			
Use original MVC view file			
Path to rendering view			
~/Views/Tabs/Tabs.cshtml			

Now you can add the rendering to appear in the toolbox in your site.

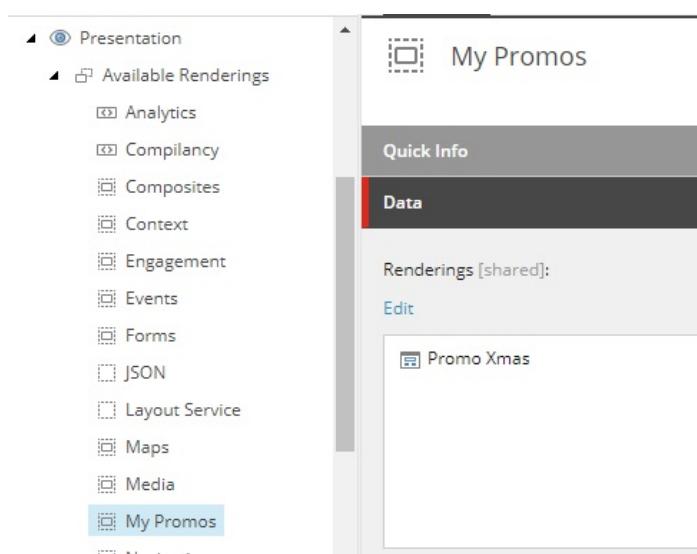
[**Add a rendering to the toolbox**](#)

To add the rendering to the toolbox:

1. Navigate to the Presentation folder of your site and right-click Available Renderings.
2. Click Insert and click Available Renderings.
3. Enter the section name for the rendering that you want to add and click

OK.

4. Click the new section that you just created, and in the Data section, click Edit.
5. In the Select Item dialog box, in the Renderings folder, click the relevant rendering, and click the right arrow to move it to the list of selected items. Click OK.



Your new rendering is now available for use in the Experience Editor.



Create a new SXA module

Abstract

Extend SXA by adding a module.

An SXA module consists of templates, branches, settings (to store scaffolding items), media library items, renderings, layouts, and so on. Default SXA modules include the rendering sections, JSON site setup, and grid systems. SXA modules are stored in the Feature or Foundation folder:

/sitecore/System/Settings/Foundation/Experience Accelerator/

/sitecore/System/Settings/Feature/Experience Accelerator/

You might want to create a new module, for example, to add a new section in the Toolbox for custom renderings, to add a base template to extend one of the site items and use custom fields, to extend a theme by adding custom assets (CSS, JavaScript), or to invoke a script during site creation.

To add a new module:

1. Navigate to sitecore\System\Settings, right-click Features or Foundation, click Insert, and then click Module.

Note

Do not place your custom modules in the Experience Accelerator folder. This folder is overwritten during updates.

2. In the Create new module dialog box, fill in the following fields:

Field	Description
-------	-------------

Module name Enter a name for the new module. This name is used as the folder name for the system areas selected.

Add to module group Select the module folder this module belongs to.

System areas Select the system areas for which container folders should be created.

Module scaffolding actions To install the new module with SXA scaffolding, select Tenant Setup and/or Site Setup.

3. Click Proceed.

Your module is now added and ready for you to customize by adding templates, branches, settings (scaffolding configurations) media library items, renderings, layouts, and so on.

Depending on which system areas you selected, there are folders in the Media Library, Templates, Branches, Renderings, Layout, and/or Layouts sections.

 Search Results (My promos Site Setup)

Subitems	 My promos Site Setup
Templates	 My promos
Branches	 My promos
System	 My promos
	 My promos Tenant Setup
Renderings	 My promos
Layout	 My promos
Layouts	 My promos
Media Library	 My promos

Data sources

Abstract

Learn more about how renderings relate to data source items.

SXA comes with a library of predefined renderings to ensure modular component based design. Most SXA renderings are designed for reusability and pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. When you add a rendering to a page, in the Associated Content dialog you can select an existing or create a new data source item.

The following fields determine how a rendering relates to its data source item:

- Datasource Location – specify where the user is allowed to look for the data source.
- Datasource Template – specify the types of data sources users can create.
- Data source – specifies a data source item.

This topic describes how SXA renderings, depending on how they use data source items, can be divided into five groups:

- [No data source item](#)
- [Optional data source item](#)
- [Single data source item per platform/tenant/site](#)
- [Reusable data source](#)
- [Select Page as a data source](#)

- [Auto-generated data source](#)

Note

When building a new rendering, the Sitecore developer must decide which category a new custom rendering falls into, so the rendering can be properly configured.

No data source item

Renderings without data source items do not store their own data, for example, the Navigation and Breadcrumb renderings. They are static and not editable by content authors.

Field	Value
-------	-------

Datasource Location Leave empty.

Datasource Template Leave empty.

Data source Leave empty.

Optional data source item

By default, renderings with an optional data source item read data from the current page, but can be set to read information from the data source item instead. For example, the Title, Subtitle, and Content renderings.

Field	Value
-------	-------

Datasource Location Leave empty

Datasource Template Leave empty

Data source Leave empty or reference a data source item

Single data source item per platform/tenant/site

Renderings with a single data source item always use one data source that is unique for a specific platform, tenant, or site. For example, the Cookie Warning and Promo renderings.

Field	Value
-------	-------

Datasource Location Leave empty

Datasource Template Leave empty

Data source Data source item reference

Reusable data source

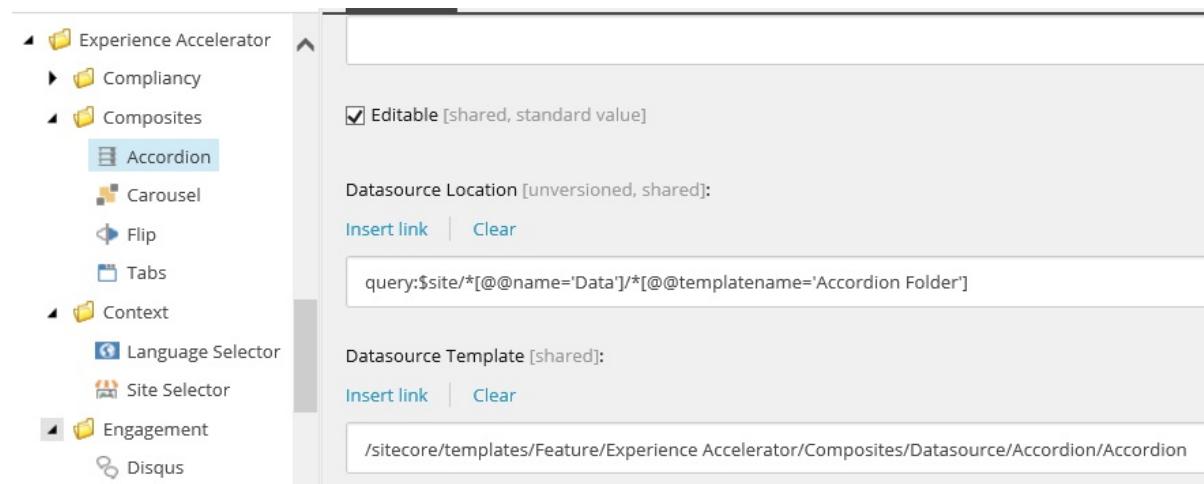
Renderings with a reusable data source enable the content editor to create and/or select a data source item. For example, the Accordion, Carousel, and Video renderings.

Field	Value
-------	-------

Datasource Location Reference to the item that groups rendering data source items.

Datasource Template Reference to the data source item template.

Data source Leave empty.



Note

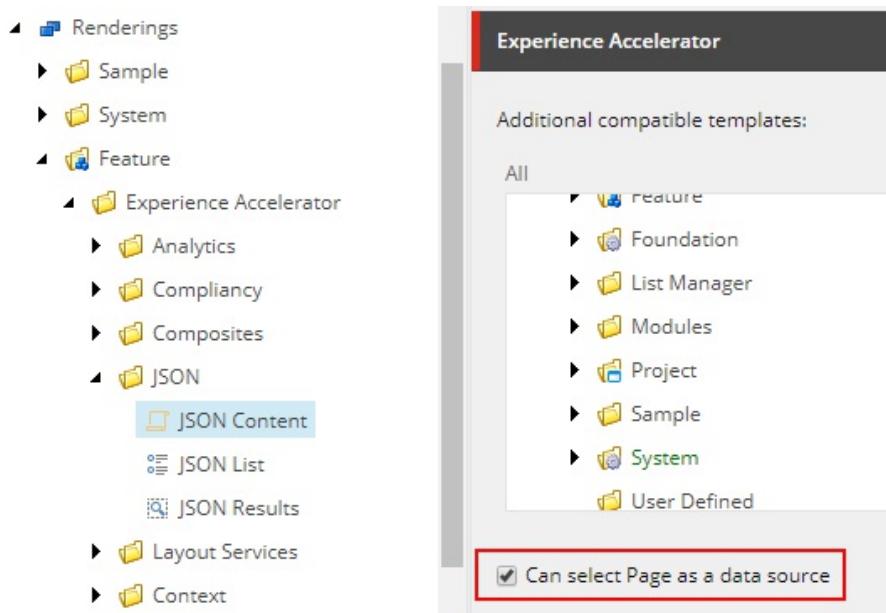
To select additional templates, navigate to the rendering `sitecore/Layout/Renderings/Feature/rendering` group/`rendering` and in the Experience Accelerator section, in the Additional compatible templates field, click the template and click the right arrow to move it to the list of selected templates.

Select Page as a data source

Renderings that can use content from the current page have an option to select the page as a data source. When the Can select Page as a data source box is selected, it lets you use a page as a data source. This option is available for renderings that take content from the current page such as Page Content,

Page List, JSON content, and JSON list.

For example, navigate to sitecore/Layout/Renderings/Feature/JSON/JSON Content and in the Experience Accelerator section select the Can select Page as a data source option.



Auto-generated data source

Renderings with an automatically generated data source store data and create the data source item when a content editor places the rendering on the page. For example, the Rich Text, Image, and Plain HTML renderings.

Field

Datasource Location Reference to the item that groups rendering data source items.

Datasource Template Reference to the data source item template.

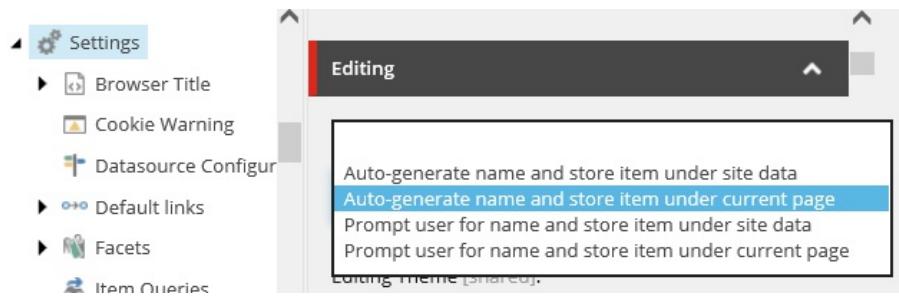
Data source Leave empty.

SXA automatically generates data items for white-listed folder terminated type and this name must be added to the configuration:

```
Additional <experienceAccelerator>
setup           <!-- List of renderings for which a data source is
                <autoDatasourceRenderings>
                  <rendering name="Image">{4A0D4E0B-BC20-4F2E-BD8
                   </autoDatasourceRenderings>
                </experienceAccelerator>
```

Administrators can configure the behavior of non-reusable components in the Editing section of the Settings item (/sitecore/content/TENANT_GROUP/TENANT/SITE/Settings) by selecting one of the following options:

- Auto-generate name and store item under site data – a new data source item is created with an auto-generated name and is stored in the location shared across the entire site.
- Auto-generate name and store item under current page – a new data source item is created with an auto-generated name and is stored as a page relative data source item under a given page.
- Prompt user for name and store item under site data – the author is prompted to provide a name for the new data source item, which is stored in the location shared across the entire site.
- Prompt user for name and store item under current page – the author is prompted to provide a name for the new data source item, which is stored as a page relative data source item under a given page.

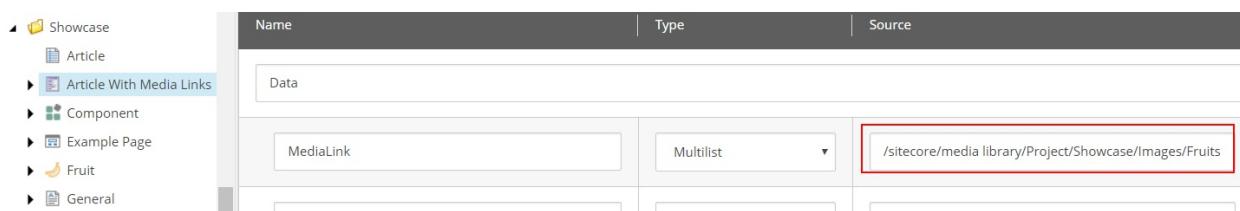


Run a source field report to set the data source context

Abstract

Use the Source field report helper script to set the data source context.

Most SXA renderings are designed for reusability and to pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. The Source field specifies where the user is allowed to look for the data source.

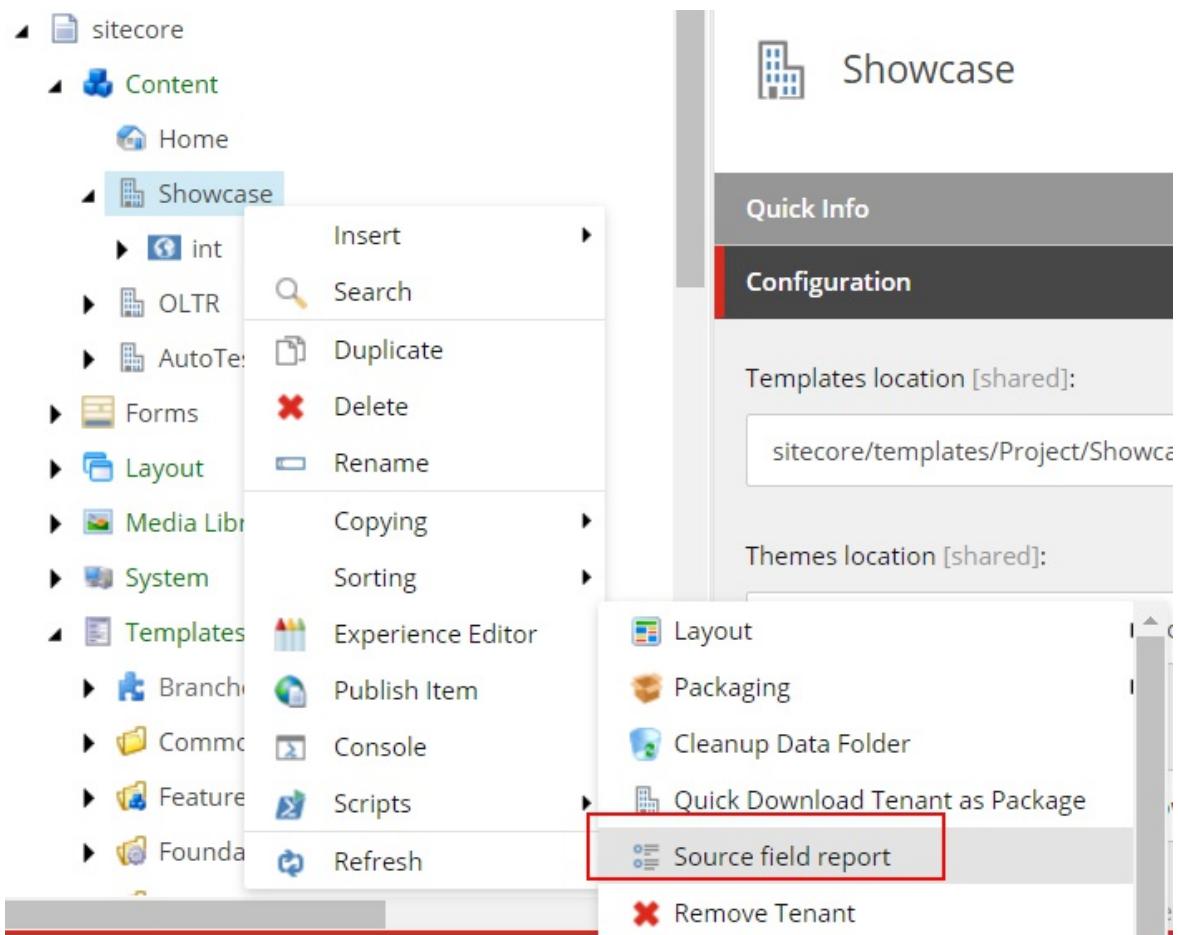


Setting this location correctly makes selecting data sources much easier because, for example, the Content Editor is directed to a specific site subfolder and does not have to search the whole Media Library for an image. You can set the location manually, use a query, or use the Source field report helper script.

To run the Source field report helper script:

1. In the Content Editor, right-click the item for which you want to invoke the Source field report. This item can be:
 - An item whose template inherits from the *Tenant* template of your tenant.
 - An item within a Project (/sitecore/templates/Project).
2. Click Scripts, and click Source field report. The script looks for the

fields that do not have a Source field specified.



3. In the Powershell Script Results dialog box, review the results. The report shows a list of all the fields that can be improved (fields with an empty Source field).

PowerShell Script Results				
Icon	Name	Path	Type	Recommendation
	Link	/sitecore/templates/Project/F/T/Product/Custom/Link	General Link	query:\$site
	Logo	/sitecore/templates/Project/F/T/Product/Custom/Logo	Image	query:\$siteMedia

4. If you think the setting is correct, click the item and click Set

recommended Source.

5. To add a different data source location manually, click the item and then click Open.

The SXA pipelines

Abstract

Understand how the SXA pipelines and their processors work.

Understanding how the SXA pipelines and their processors work provides you with an insight into how dependencies are rendered, how tokens for rendering variants are created, how CSS classes are generated, and so on.

A pipeline consists of a sequence of processors. A processor is a .NET class that implements a method. When a pipeline is invoked, the processors are run in order. You can extend and customize the pipelines by adding or replacing processors. Extending a pipeline involves modifying the pipeline definition located in a Sitecore patch file.

This topic describes the following pipelines:

- [resolveVariantTokens](#)
- [loc](#)
- [decoratePage](#)
- [decorateRendering](#)
- [mediaRequestHandler](#)
- [resolveTokens](#)
- [assetService](#)
- [getControlEditability](#)
- [getRobotsContent](#)

- [refreshHttpRoutes](#)
- [getVelocityTemplateRenderers](#)
- [resolveSearchQueryTokens](#)
- [resolveBoostingQuery](#)

[resolveVariantTokens](#)

The `resolveVariantTokens` pipeline is used to create tokens for rendering variants.

This pipeline includes the following processors:

Processor	Description
ResolveIFileTypeIcon	Renders the span HTML element with the class according to the extension of the file.
ResolveItemId	Specifies the ID of the content item to be resolved.
ResolveItemName	Specifies the name of the content item to be resolved.
ResolveSize	Renders the file size.

[ioc](#)

The Inversion of Control (IoC) design principle allows you to change rendering dependencies without changing the rendering itself. The `ioc` pipeline is defined in the `Sitecore.XA.Foundation.IoC.config` file and is

used for adding processors in which you can register your custom services in the container.

For example, you can add the RegisterPageContentServices processor to register services used in the Page Content feature.

```
<pipelines>
  <ioc>
    <processor type="Sitecore.XA.Feature.PageContent.Pipeline"
    </ioc>
</pipelines>
```

[decoratePage](#)

The decoratePage pipeline is used to decorate the page `<body>` tag with attributes. These can be standard attributes, such as `id` or `class`, but you can also add custom data attributes.

[decorateRendering](#)

The decorateRendering pipeline is used to decorate the rendering's outer `<div>` tag with attributes. These can be standard attributes, such as `id` or `class`, but you can also add custom data attributes.

```
<div class="component title">
<div class="component-content">
<h2 class="field-title">
Title of the page
</h2>
</div>
</div>
```

[mediaRequestHandler](#)

The mediaRequestHandler pipeline is used in the SXA media requests handler. The mediaRequestHandler pipeline is defined in the `Sitecore.XA.Foundation.MediaRequestHandler.config` file.

This file extends the standard media request handler by adding a pipeline that

implements custom functionalities such as the support of wireframe images or providing optimized assets.

This pipeline includes the following processors:

Processor	Description
ParseMediaRequest	Checks if the HTTP request is a valid media type. If not, the pipeline is aborted.
GetMediaFromUrl	Determines the media item from the URL that is stored in HTTP request.
HandleErrors	Redirects the user to an error page.

[resolveTokens](#)

The `resolveTokens` pipeline is used to resolve tokens that can be used in queries. For example:

- `$compatibleThemes` - path to all themes
- `$theme` - currently used theme
- `$pageDesigns` - root of page designs
- `$partialDesigns` - root of partial designs
- `$currenttemplate` - name of the current template
- `$tenant` - path to the current tenant
- `$site` - path to the current site

- \$home - path to the current site start item (Home)
- \$templates - path to the current site templates
- \$siteMedia - path to Virtual Media folder located under site

By adding new processors to this pipeline, you can design new tokens.

This pipeline is defined in the `Sitecore.XA.Foundation.TokenResolution.config` file and includes the following processors:

Processor	Description
CurrentTemplateToken	Determines the current tokens used.
EscapeQueryTokens	Used to escape tokens that are used in Sitecore queries.

[**assetService**](#)

The `assetService` pipeline is responsible for assets optimization. This pipeline includes the `AddEditingTheme` processor, which you can use to add a theme when you are in Edit mode.

```
<assetService>
    <processor type="Sitecore.XA.Foundation.Editing.Pipelines
    </assetService>
```

[**getControlEditability**](#)

The `getControlEditability` pipeline is used to check if a rendering should be editable or not. It is defined in the `Sitecore.XA.Foundation.Editing.config` file. Used in `Editing`,

Composites, LocalDatasources.

```
<getControlEditability>
    <processor type="Sitecore.XA.Foundation.Editing.Pipelines
    </getControlEditability>
```

It contains the `IsRenderingInCurrentItemRenderings` processor that checks the component XML to see if it contains other component (by unique ID).

getRobotsContent

The `getRobotsContent` pipeline is used to extend the response provided to search crawler robots in the `robots.txt` file. The `Robots.txt` file is a simple text file on your site's root directory that tells search engine robots what to crawl and what not to crawl on your site. The `getRobotsContent` pipeline contains the following processors:

Processor	Description
GetContentFromSettings	Checks if the robots' content field is filled and uses its value.
GetDefaultRobotsContent	Checks the <code>robots.txt</code> file for a value when the robots' content field is empty.
AppendSitemapUrl	Adds the path of the <code>sitemap.xml</code> file to the robots' content field.

getRenderingCssClasses

The `getRenderingCssClasses` pipeline is used to gather CSS classes that will be applied on rendering (added to the list of CSS classes on rendering).

[refreshHttpRoutes](#)

The refreshHttpRoutes pipeline is used to refresh HTTP routes after changes in site configuration.

[getVelocityTemplateRenderers](#)

The getVelocityTemplateRenderers pipeline is used to add a custom renderer that later on can be used in the NVelocity template. This pipeline is defined in the Sitecore.XA.Foundation.RenderingVariants.config file. It contains the following processors:

Processor	Description
InitializeVelocityContext	Initializes the pipeline argument objects.
AddTemplateRenderers	Provides two custom tools that format data and time values (dateTool) and numbers (numberTool) in an NVelocity template.

[resolveSearchQueryTokens](#)

The resolveSearchQueryTokens pipeline is used to add search filters. This pipeline is defined in the Sitecore.XA.Foundation.Search.config file. By default, the following tokens are available:

- TaggedTheSameAsCurrentPage|SxaTags
- TaggedWithAtLeastOneTagFromCurrentPage|SxaTags
- UnderCurrentPage

- ExcludeCurrentPage
- ItemsOfTheSameTemplateAsTheCurrentPage
- ItemsWithTheSameValueInField|FieldName

resolveBoostingQuery

The resolveBoostingQuery pipeline is used to add extension point that can be used to write your own search boosting rules and then use them in SXA. To do that you can write your own custom rule and use the resolveBoostingQuery pipeline to turn it into a query. This pipeline is defined in Sitecore.XA.Foundation.Search.config file and includes the following processor:

```
resolveBoostingQuery
    processor type="Sitecore.XA.Foundation.Search.Pipelines.Resol
    /resolveBoostingQuery
```

The SXA script library

Abstract

SXA includes a number of scripts to help with the most common tasks.

SXA includes a number of PowerShell scripts to automate the most common tasks. The [Sitecore PowerShell Extensions \(SPE\) module](#) provides a command line and a scripting environment and enables you to use PowerShell from within Sitecore. In this way, you can run commands and write scripts according to Windows PowerShell syntax. Every SXA module that uses SPE has its own script library in:
`sitecore\System\Modules\PowerShell\Script Library`.

The naming convention for these scripts is: SXA - Module name:

- ▶ SXA - Compliancy
- ◀ SXA - Creative Exchange
 - ◀ Functions
 - ☒ ResolveCreativeExchangePathTokens
- ▶ SXA - Forms
- ▶ SXA - Geospatial
- ▶ SXA - LocalDatasources
- ▶ SXA - Multisite
- ▶ SXA - Search
- ▶ SXA - Site Metadata
- ...

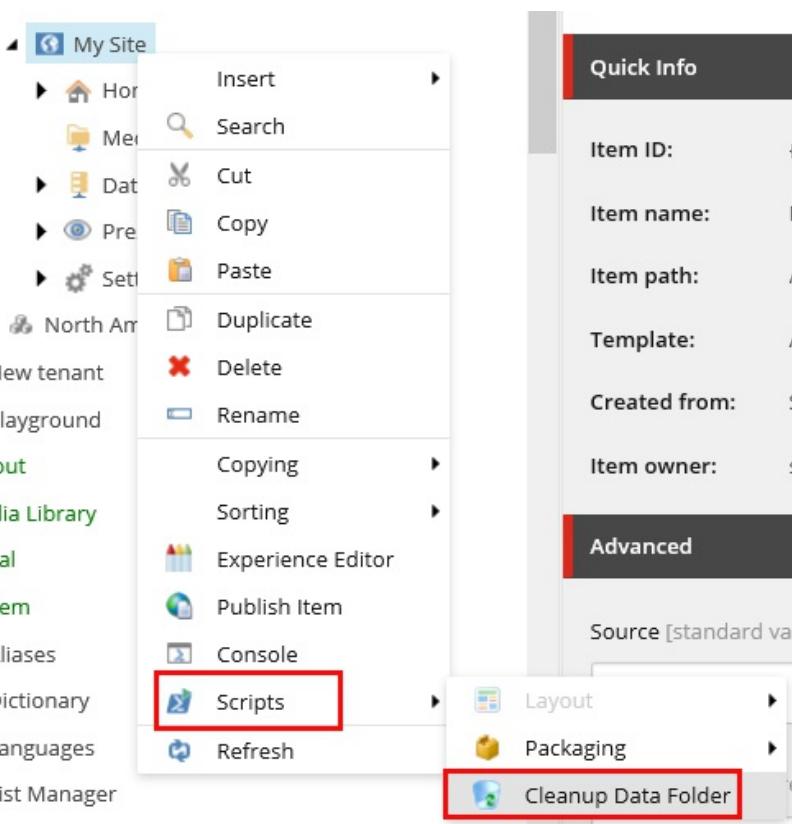
Note

PowerShell scripts can be used to automate tasks that you find yourself doing on a regular basis. To write your own scripts, or view code of existing scripts, use the PowerShell Integrated Scripting Environment (ISE). You can access this tool on the Sitecore Launchpad:



The types of SXA scripts vary from cleanup scripts to cmdlets that add insert options to items:

- Scaffolding – scripts used during scaffolding to automate the process of site/tenant creation.
- Context Menu – scripts that are available for editors using the Content Editor. For example, the cleanup data sources script:



- Cmdlets – a lightweight command that is used in SPE and can be reused by other developers.

- Insert Item – scripts that extend the Insert section in the Content Editor.

The following table describes the available SXA script modules, their functions, descriptions, and type of script:

Module	Function	Description
SXA – Compliancy	AddCookieWarning	Inserts the Cookie Warning partial design.
SXA – Creative Exchange	ResolveCreativeExchangePathTokens	Resolves the path to the Exchange FileStorage item to enable unique folder. This prevents overwriting the folder.
SXA – Forms	Set forms folder	Sets the WFFM Form definition item. This creates its own forms root folder under WFFM – there is one folder).
SXA – Geospatial	Assign POI items	Assigns a default rendering type. This enables each same rendering variant.
SXA – LocalDatasources	Cleanup Data Folder	Searches and cleans the Page Data folder.
		Decorates new Tenant.

	Add Content fields	additional fields. Add to a Page template.
SXA – Multisite	Home base template	Adds a base template under Tenant template as a base template.
	Set home item title field	Sets the home item title.
	Set start item	Sets the start item or By default, the start item is the root of all pages.
SXA – Search	Set default sorting facet	Assigns the Title facet to sort data sources.
	AddBrowserTitle	Inserts the BrowserTitle partial design.
	AddCustomMetadata	Inserts the CustomMetadata partial design.
	AddFavicon	Inserts the Favicon partial design.
	AddOpenGraphMetadata	Inserts the OpenGraphMetadata partial design.
SXA – Site Metadata		Inserts the SeoMetadata partial design.

AddSeoMetadata	The PowerShell wrapper for <code>IMultiSiteContext.AddSeoMetadata()</code> . Resolves the Seo metadata partial design.
AddStickyNotes	The PowerShell wrapper for <code>IMultiSiteContext.AddStickyNotes()</code> . Resolves the Sticky Notes partial design.
AddTwitterMetadata	The PowerShell wrapper for <code>IMultiSiteContext.AddTwitterMetadata()</code> . Resolves the Twitter Metadata partial design.
AddViewport	The PowerShell wrapper for <code>IMultiSiteContext.AddViewport()</code> . Resolves the Viewport partial design.
MultisiteContext/Get-DataItem	The PowerShell wrapper for <code>IMultiSiteContext.GetDataItem()</code> . Resolves the Data Item.
MultisiteContext/Get-SettingsItem	The PowerShell wrapper for <code>IMultiSiteContext.GetSettingsItem()</code> . Resolves the Settings Item.
MultisiteContext/Get-SiteItem	The PowerShell wrapper for <code>IMultiSiteContext.GetSiteItem()</code> . Resolves the Site Item.
MultisiteContext/Get-SiteMediaItem	The PowerShell wrapper for <code>IMultiSiteContext.GetSiteMediaItem()</code> . Resolves the Virtual Media Item of the current site.

	MultisiteContext/Get-TenantItem	The PowerShell wrapper cmdlet for the <code>IMultiSiteContext</code> interface. Resolves the Tenant.
	PresentationContext/Get-PageDesignsItem	The PowerShell wrapper cmdlet for the <code>IPresentationContext</code> interface. Resolves the Page Design item for the current site.
	PresentationContext/Get-PartialDesignsItem	The PowerShell wrapper cmdlet for the <code>IPresentationContext</code> interface. Resolves the Partial Design item for the current site.
SXA – Scaffolding	Add-BaseTemplate	Cmdlet that adds a base template to the current item.
	Add-FolderStructure	Cmdlet that recreates the folder structure if it does not exist.
	Add-InsertOptionsToItem	Cmdlet that adds <code>InsertOptions</code> to an item.
	Add-InsertOptionsToTemplate	Cmdlet that adds <code>InsertOptions</code> to a Template item.
	Get-PartialDesign	Returns the first Partial Design item.

	Designs folder.
Test-ItemIsPartialDesign	Checks whether the PartialDesignTemp
Test-ItemIsSiteDefinition	Checks whether the SiteDefinitionTem
New-Site	Contains all the func site.
New-Tenant	Contains all the func tenant.
Site	<p>Adds script to the In menu that lets users when the following is true:</p> <ul style="list-style-type: none"> Rule 1 where the item type is Site Rule 2 where the item type is Site
Tenant	<p>Adds script to the In menu that lets users when the following is true:</p> <ul style="list-style-type: none"> Rule 1 where the itemID is Tenant Rule 2 where the item type is Tenant

Walkthrough: Building a simple rendering

The SXA toolbox contains default renderings for simple text, images, videos, social media plugins, and so on. You can also create a new rendering. This walkthrough describes how to:

- [Create a rendering](#)
- [Create the controller and action method](#)
- [Inject a repository into a controller](#)
- [Create a controller rendering in Sitecore](#)
- [Register dependencies](#)
- [Add grid and styling support](#)
- [Add a new rendering to the toolbox](#)

[Create a rendering](#)

SXA renderings are Sitecore controller renderings and SXA uses the same basic structure for all renderings. The wrappers are the same for every rendering and always contain:

- The component element
- The `Html.Sxa().Component()` helper
- The content: `div class="component-content"`

Note

To be able to use the Creative Exchange functionality in SXA, you must use the `Html.Sxa().Component()`. This helper injects data attributes for a Creative Exchange export.

To create a rendering:

- In the rendering definition item, specify what action Sitecore takes to render the component. For example, for the title rendering:

```
<div @Html.Sxa().Component("title", Model.Attributes)>
    <div class="component-content">
        @foreach (VariantFieldBase variantField in Model.VariantFields)
        {
            @Html.RenderingVariants().RenderVariant(variantField, Model.Item, Model.RenderingWebEditingParams)
        }
    </div>
</div>
```

Create the controller and action method

Every SXA rendering needs a controller. SXA enables you to inherit from a base SXA controller. It provides useful properties and logic to automatically resolve the view name and provide the model (*Rendering*, *PageContext*). It is best practice to use these base classes when you build a new rendering or extend an existing rendering.

To create the controller and the action method:

1. In an ASP.NET MVC Web application project in Visual Studio, create a new controller. In SXA, controllers are kept fairly simple and most of the logic is implemented in repositories. To make the controllers use the repositories, you must create controller renderings with a construction injection.
2. Use the `StandardController` base class to implement the standard `Index()` action method that automatically resolves the view name based on the rendering item name. For example:

```
public class SimpleComponentController: StandardController
{
    private readonly ISimpleComponentRepository _reposito
```

```

public SimpleComponentController(ISimpleComponentRepo
{
    _repository = repository;
}

protected override object GetModel()
{
    return _repository.GetModel();
}
}

```

Inject a repository into a controller

SXA uses repositories to provide the model for the MVC views. Therefore, when you finish the controller, you must inject the repository into the controller. You can inject your repository in two ways:

- Construction injection - we recommend to use this injection. For example:

```

public class SimpleComponentController : StandardController
{
    private readonly ISimpleComponentRepository _repository;

    public SimpleComponentController(ISimpleComponentReposito
    {
        _repository = repository;
    }

    ...
}

```

- ServiceLocation injection - this injection is used in processors or commands and sometimes in controllers. For example:

```

public class SimpleComponentController
{
    private readonly ISimpleComponentRepository _repository;

    public SimpleComponentController()
    {
        _repository = ServiceLocator.ServiceProvider.GetService<ISimpleComponentRepository>();
    }
}

```

```
    }
```

```
    ...
```

SXA provides base repositories. For example, the VariantsRepository and the ListRepository provide useful properties and make implementing your own repository a lot easier. Another example is the ModelRepository base class that provides useful properties such as:

- **PageContext** - represents information required to service an individual request for a specific device, containing information about the HTTP response under construction. The PageContext contains properties that expose the requested item, device, and the PageDefinition object created for the request.
- **IsEdit** - checks if Edit mode is on.
- **IsControlEditable** - specifies whether the rendering should be editable.
- **Attributes** – dictionary of attributes that will be applied to the rendering.

All repositories must have their own interface, even when a repository is empty and is just using base interface methods. This is because it is used later on when registering the Dependency Injection container:

```
public interface ISimpleComponentRepository : IModelRepository
{
}
```

And here is an example of simple repository:

```
public class SimpleComponentRepository : ModelRepository, ISI
{
    public override IRenderingModelBase GetModel()
    {
        SimpleComponentModel model = new SimpleComponentModel

        FillBaseProperties(model);
        model.Title = GetTitle();

        return model;
    }
}
```

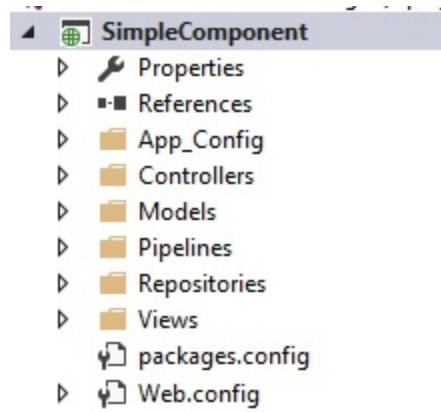
```

        }

        private string GetTitle()
        {
            return PageContext.Current[Templates._Title.Fields.Ti
        }
    }
}

```

Remember to register your repository in the DI container. You can do that using a config file:



And the view could look like this:

```

@model Sitecore.SXA.Startup.Feature.SimpleComponent.Models.Simple


You are on @Model.Title page/h1
    /div
/div


```

Note

Most SXA renderings are designed for reusability and pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. You can add data source logic to the rendering by using:

```
if (DataSourceItem == null) { ... }
```

Create a controller rendering in Sitecore

To create the rendering for users to access from the Toolbox, you have to create the item in Sitecore and use the controller name and the action name to link the item with the code.

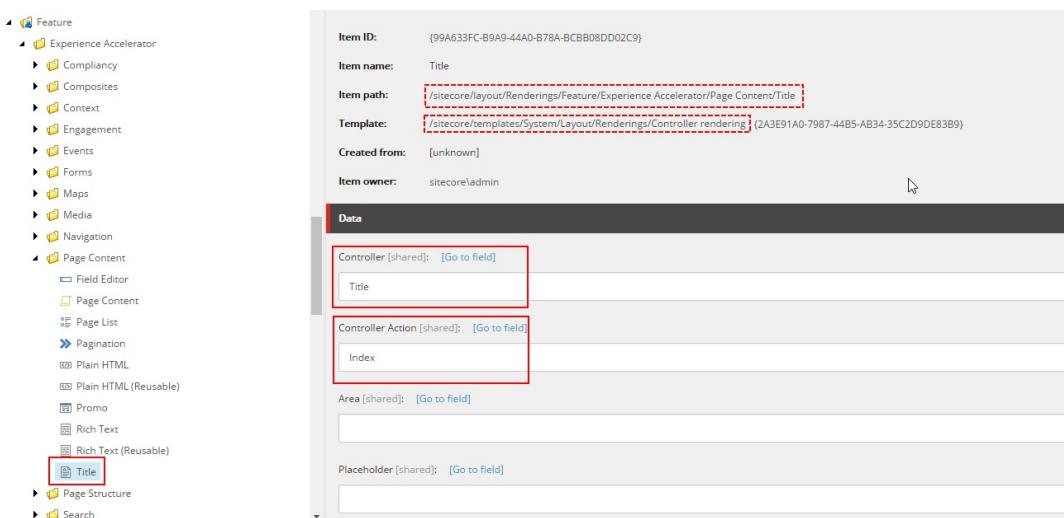
To create a controller rendering in Sitecore:

1. In the Content Editor, go to `sitecore/Layout/Renderings/Feature/Experience Accelerator` and right-click the section that you want to add the rendering to.

Note

The Section names in `Layouts/Renderings/Feature/Experience Accelerator` are the actual names used in the Toolbox.

2. Click Insert, Controller Rendering. For example, to add the controller rendering item for the Title rendering in:
 - The Controller field – enter Title
 - The Controller Action field – enter Index. This action is inherited from the StandardController.



- In the Experience Accelerator section, you can add additional settings for your rendering. For example, a specific CSS class, whether it can use a page as a datasource or whether it can use dynamic placeholders:

Field	Description
Additional compatible templates	Enables to select multiple templates for data source. For example, for the Promo rendering you use the Promo templates but also other data sources such as Text and Image.
Can select Page as a data source	Renderings that can use content from the current page have an option to select the page as a data source. When the Can select Page as a data source box is selected, it lets you use a page as a data source. This option is available for renderings that take content from the current page such as Page Content, Page List, JSON content, and JSON list.

Enter the CSS class for the rendering. For example:

Rendering css class

Rendering css class: [\[Go to field\]](#) | [\[Reset Field\]](#)

```
▼ <div class="component custom-promo col-xs-12">
  <div class="component-content">...</div>
```

Rendering view path

By default, the view name in SXA is resolved as the rendering definition item name. To provide a custom path, enter the location of the view here.

Specify any other properties here. For example:

- `IsRowSplitterRendering` – for renderings that

split rows.

- `IsColumnSplitterRendering` – for renderings that split columns.
 - `IsRenderingsWithDynamicPlaceholders` – for renderings that use dynamic placeholders.
- OtherProperties
- `IsPaginationEnabledRendering` – for renderings that support pagination.
 - `IsNeverLocalRendering` - for disabling the editing of inherited renderings.
 - `IsCompositeRendering` – for composite renderings.
 - `IsAutoDatasourceRendering` - for renderings for which a data source is automatically created after adding that rendering to a page.

4. Save and publish the item.

Register dependencies

Dependency injection (DI) is a technique for achieving loose coupling between objects and their collaborators, or dependencies you can use the Sitecore dependence injection. Without dependency injection, a class must directly instantiate collaborator or use static references. With dependency injection, the objects a class need are provided for it. Most often, classes will declare their dependencies through their constructor. Sitecore only supports this approach, known as constructor injection. The Sitecore implementation of DI is based on the `Microsoft.Extensions.DependencyInjection` abstractions from ASP.NET Core.

Remember to register your repository in the DI container. You can do that

using a config file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
    <sitecore>
        <services>
            <register serviceType="YOUR_NAMESPACE.ISimpleComponen
                           implementationType="YOUR_NAMESPACE.SimpleCo
                           lifetime="Singleton"/>
        </services>
    </sitecore>
</configuration>
```

For SXA renderings, you can also use the Sitecore base class `IServicesConfigurator` to configure services from code:

```
using Microsoft.Extensions.DependencyInjection;
using PageTitle.Controllers;
using PageTitle.Repositories;
using Sitecore.DependencyInjection;

namespace PageTitle
{
    public class RegisterDependencies : IServicesConfigurator
    {
        public void Configure(IServiceCollection serviceCollectio
        {
            serviceCollection.AddTransient<ISimpleComponentReposi
            serviceCollection.AddTransient<SimpleComponentControl
        }
    }
}
```

Next, you need to add the config with a configurator node that points to that `RegisterDependencies` class.

For example:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
    <sitecore>
        <services>
            <configurator type="SimpleComponent.RegisterDependencies, S
        </services>
    </sitecore>
</configuration>
```

Add grid and styling support

The default SXA renderings come with grid and style support. If you want your custom rendering to be compliant with SXA grid and styling, you need to provide a parameter template and assign some base SXA templates.

To add grid and styling support for a rendering:

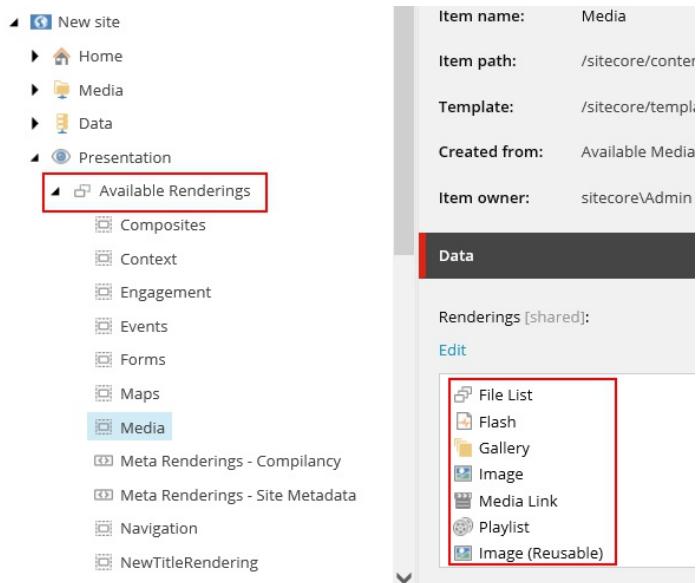
1. First, add a parameters template to your rendering. Navigate to `sitecore/Layout/Feature/Experience Accelerator` and click the new rendering.
2. In the Editors option section, in the Parameters Template field, enter the path to your template.
3. Now you can assign some base SXA templates to your rendering parameter template such as for example:
 - `IStyling` – for styles support. This template adds a new field in the control properties for that rendering and a additional button on the edit frame of the rendering.
 - `IComponentVariant` – for rendering variant support. This template adds a rendering variant drop-down in the control properties and Variant Selector drop-down on the edit frame.
 - `Grid Parameters` – for grid support. This template adds grid field in the control properties and additional button on the edit frame of the rendering.

Add a rendering to the toolbox

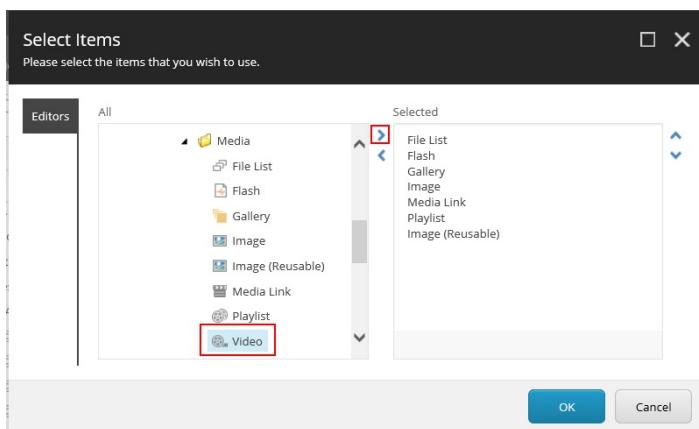
In the Presentation folder of your site, you can set which renderings are visible in the Toolbox in the Experience Editor. You can also [restructure the toolbox](#).

To add a rendering to the SXA toolbox:

1. In your site, navigate to *Presentation/Available Renderings*



2. In the Data section, in the Renderings field, click Edit.
3. In the All section on the left, click a rendering and then click the right arrow to move it to the list of selected renderings. When you have added all the renderings that you want in the SXA toolbox, click OK.



Customize rendering HTML per site

Abstract

Override the component view path.

All SXA renderings come with default HTML views. If you want to change the HTML for a rendering, SXA lets you specify site-specific views for renderings. For example, if you have a site that uses the Promo rendering a lot, but you don't like the HTML view, you can create a custom Promo rendering with a different HTML view. SXA will try to find overridden views under the specified path and if a site-specific view file exists, this file will be rendered. Otherwise, the functionality will fallback to the view from default views location.

To create the site-specific view folder:

1. Navigate to `C:\inetpub\wwwroot\Site\Website\` and create a new folder to store the site-specific views. For example, create a *Custom Views* folder for a customized component HTML for the Promo rendering: `C:\inetpub\wwwroot\Site\Website\MyCustomViews\`
2. Your new folder must have the same structure as in the root folder. Therefore, you must add a *Views* folder:
`C:\inetpub\wwwroot\Site\Website\MyCustomViews\Views\`
3. If, for example, you want to customize the Promo component, go to `C:\inetpub\wwwroot\Site\Website\Views`, copy the *Variants* folder and paste it in the *Views* folder you just created:
`C:\inetpub\wwwroot\Site\Website\MyCustomViews\Views\Variants\`
4. Navigate to `C:\inetpub\wwwroot\Site\Website\Views`, copy the *web.config* file and paste it in the folder you just created.

5. Modify the HTML.
6. In the Content Editor, navigate to sitecore/Content/Tenant/Site/Settings and in the Site Configuration section, in the CustomRenderingViewPath field specify the relative path to the folder that you just created.

Embed HTML in SXA

To help you add specific HTML to pages, two components come with SXA: the Plain HTML and Plain HTML (reusable) component and the HTML Snippet component.

You use the Plain HTML and Plain HTML (reusable) component to embed (and store) HTML code for visual elements that you want to place on your page, for example, to embed a YouTube video. This component adds the default wrapper that SXA uses for all components.

You use the generic HTML Snippet component to put on meta partial designs to make HTML elements work in the background. It lets you integrate with an external system. For example, to integrate Google Tag Manager or customer server chat functionality. This component comes without any extra surrounding markup.

The topic describes how to:

- [Add the Plain HTML component](#)
- [Add an HTML snippet](#)

Add the Plain HTML component

Plain HTML is a generic component that lets you include arbitrary static HTML blocks. You can insert the Plain HTML (reusable) component for single use as well as for reuse on multiple pages across the whole site.

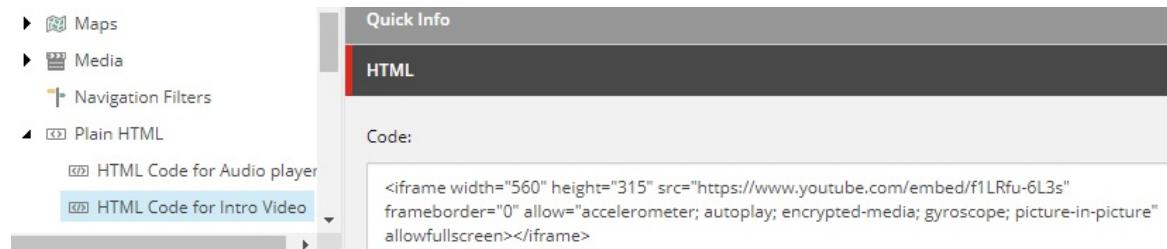
To add the Plain HTML component:

1. In the Experience Editor, navigate to the page (design) or partial design you want to add the HTML to. If you do not need the HTML to display across multiple pages, you can also add it to a page.

2. Open the Toolbox. In the Page Content group, click Plain HTML (reusable) and drag it to the page.
3. In the Select the Associated Content dialog box, click Create.
4. In the Insert Item dialog box, click Insert HTML Code.
5. Save the page.
6. In the Plain HTML (Reusable) toolbar, click Edit HTML source.



7. Your HTML Code item now opens in the Content Editor. In the HTML section, in the Code field, enter your HTML code.



Note

For complex components, you might consider using the Plain HTML rendering because it provides complete freedom regarding the HTML you can place in it. However, this way of adding content to a page can be difficult for your editors to work with because not all content editors know how to edit the HTML code. Consider using clones, variants, or the HTML snippet instead of the Plain HTML rendering.

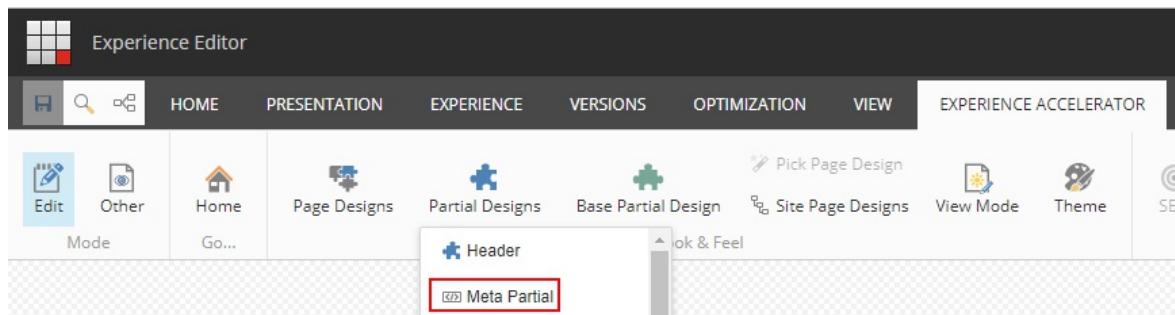
[Add an HTML Snippet](#)

The HTML Snippet is a meta component that you can add to the Metadata Partial Design. It can be used to add custom HTML to the <head> section,

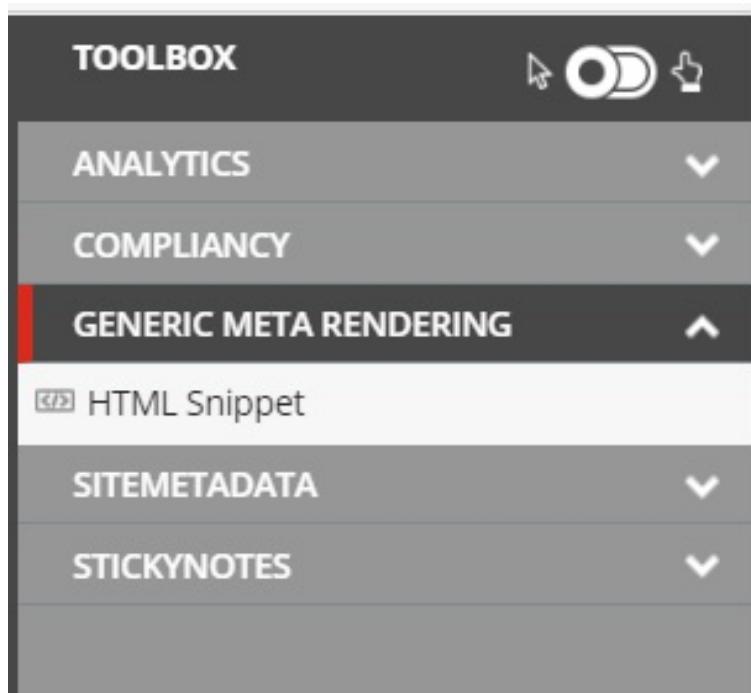
directly after the <body> tag or before the closing </body> tag of each page. This can be convenient when you want to embed a third-party component (widget, tracking script, and so on), or when none of the available components suit your custom needs.

To add an HTML snippet:

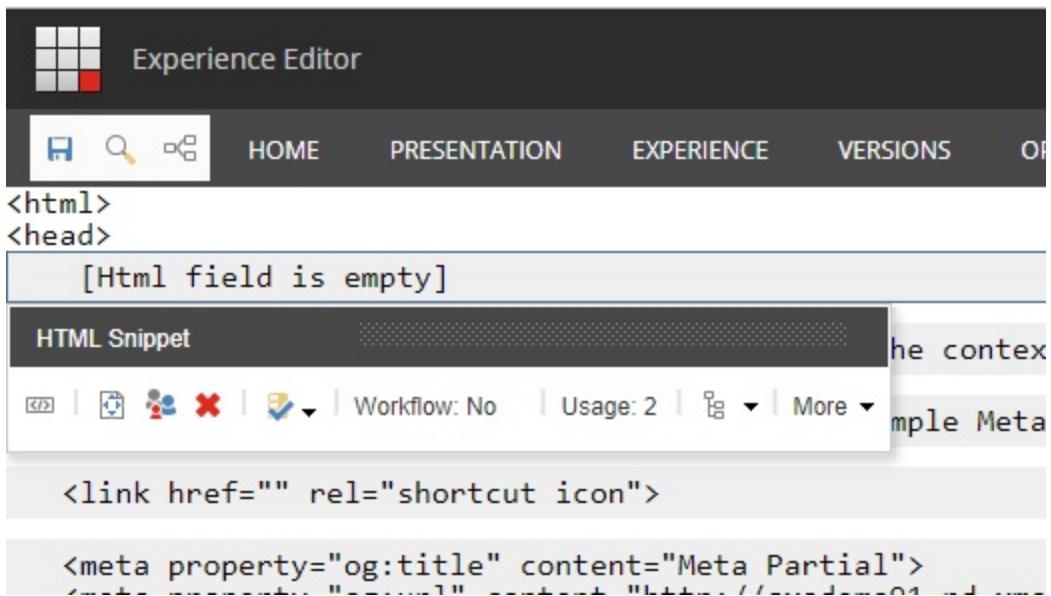
1. In the Experience Editor, click Partial Design and click Meta Partial.



2. In the SXA Toolbox, in the Generic Meta Rendering category, drag and drop the HTML Snippet component. You can add meta renderings to the following containers: to the head section, directly after the body tag, or before the closing /body tag of each page.



If, for example, you want to integrate Google Tag Manager, it is important to add it to the head.



3. In the Select the Associated Content dialog box, click Create.
4. In the Insert Item dialog box, click HTML snippet.
5. Save the page.
6. In the HTML snippet toolbar, click Edit HTML source.
7. Your HTML Code item now opens in the Content Editor. In the HTML Snippet section, in the html field, enter your HTML code.

Create a custom token for a rendering variant

SXA lets you change the way components render by using [rendering variants](#). If you quickly want to extend a rendering variant, you can use the custom `resolveVariantTokens` pipeline to extend the set of variant tokens.

A custom token can help you format certain field values and expose more complex data that is not available in normal rendering variant items, for example, because you want to do a transformation on the values of Sitecore items. This can be convenient when, for example, you want to display the parent item field in a rendering variant of the PageList component.

To create a custom token for a rendering variant:

1. Define your own token by adding an additional processor to the `resolveVariantTokens` pipeline. First, create the token processor. For example, `$parentName`:

```
public class ResolveParentName : ResolveVariantTokensProcessor
{
    public override string Token => "$parentname";

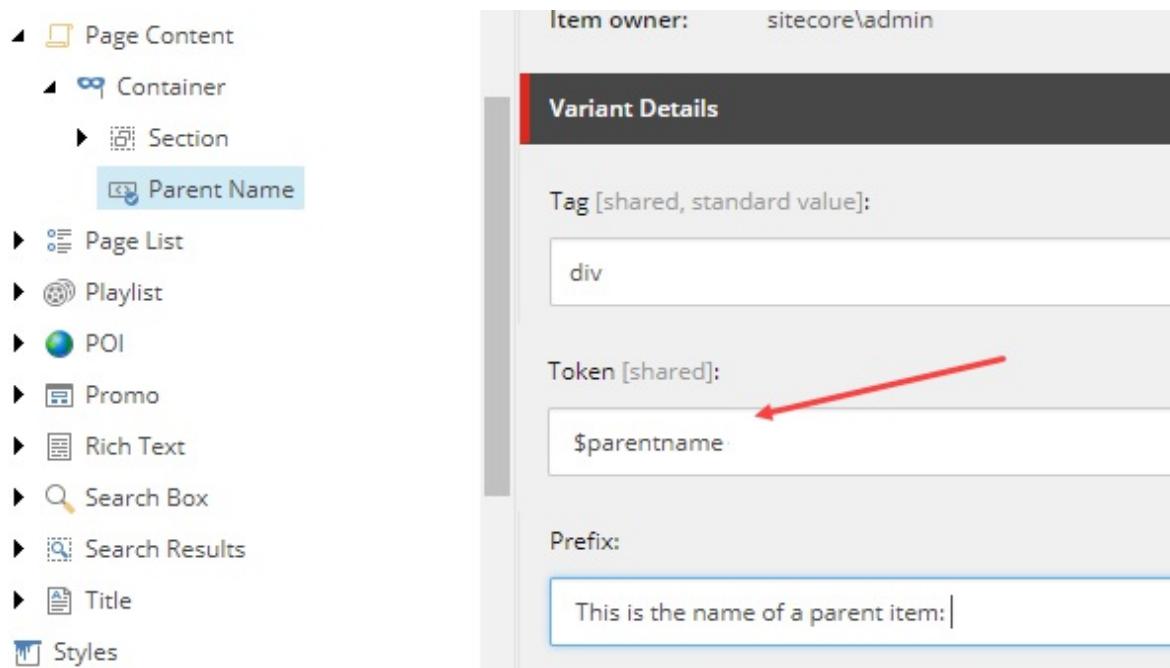
    public override void ResolveToken(ResolveVariantTokensArgs
    {
        // this if statement is quite important in order to s
        if (args.ResultControl != null)
        {
            args.ResultControl.Controls.Add(new LiteralControl
            {
                Text = args.ContextItem.Parent.Name
            });
        }
        else
        {
            args.Result = args.ContextItem.Parent.Name;
        }
    }
}
```

}

2. Register the processor in the `resolveVariantTokens` pipeline:

```
pipelines
    <resolveVariantTokens>
        <processor type="YOUR_NAMESPACE.ResolveParentName, YO
        </resolveVariantTokens>
    </pipelines>
```

3. In the Content Editor, navigate to `sitecore/Content/Tenant/Site/data/Content` token to change an existing token or to insert a new one.
4. To apply the token, in the Content Editor, navigate to the rendering variant that you want to add the token to and, in the Token field, enter the custom token using the `$` key.



Add a data source setting for an existing rendering

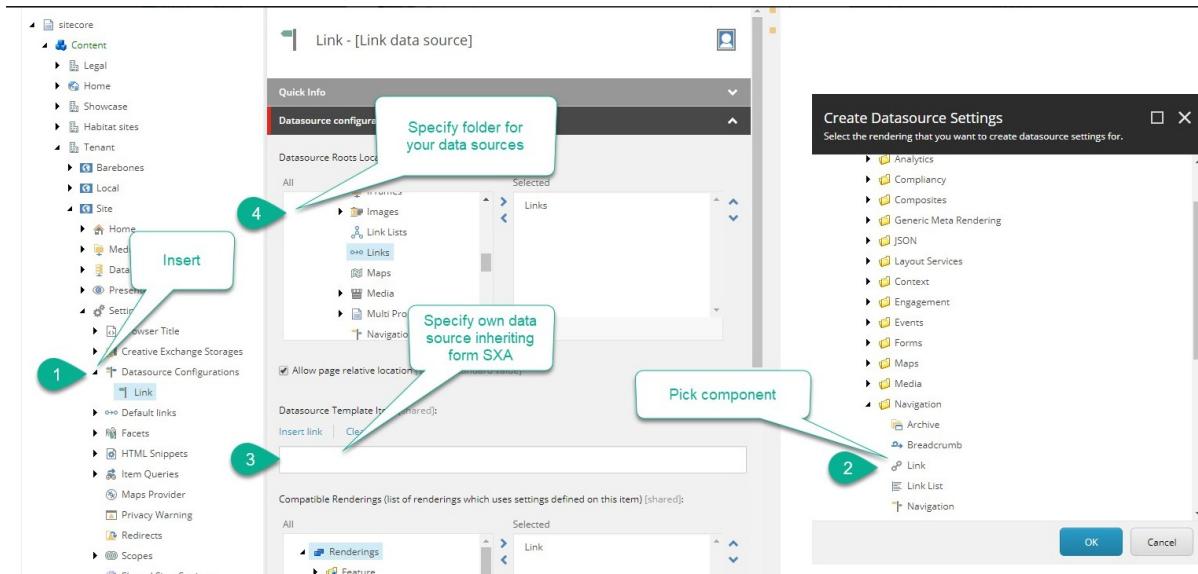
Most SXA renderings are designed for reusability and pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. Sometimes, you might want to add a field to the data source item of a rendering. For example, because you want to add a template field Image to the data source item of the Link rendering.

To add a data source setting to a rendering:

1. First, [create the new template](#). In the Content Editor, navigate to your tenant templates folder (`sitecore/Templates/Project/`) and insert the new template. This new data source item template must inherit from the existing SXA data source item template.
2. In the Content Editor, navigate to `sitecore/content/Tenant/Site/Settings/`, right-click Datasource Configurations, click Insert, and click Datasource configuration.
3. In the Create Datasource Settings dialog box, select the rendering that you want to create the data source setting for.
4. In the Datasource configuration section, in the Datasource Template Item field, insert the new template.
5. In the Datasource configuration section, in the Datasource Roots Locations field, specify the folder that contains the data sources.

Note

Consider creating a data source folder with your own insert options so that your editors can create them easily.



Grid

By default, SXA comes with three grid systems. If you do not want to use the bootstrap, foundation, or Grid960 grid system, you can [add your own custom CSS framework](#).

Create a custom grid

Abstract

Add your own CSS framework.

By default, SXA comes with four grid systems. If you do not want to use the bootstrap (3 or 4), foundation, or Grid960 grid system, you can add your own custom CSS framework.

Grid systems are stored as a feature in Sitecore and include:

- The grid theme – includes the css file with all grid classes.
- The grid definition item – includes all grid-system specific items (the devices and their classes).
- The grid setup item – the scaffolding item to include the grid in the drop-down list of the site creation wizard.



This topic describes how to:

- [Create a grid theme](#)

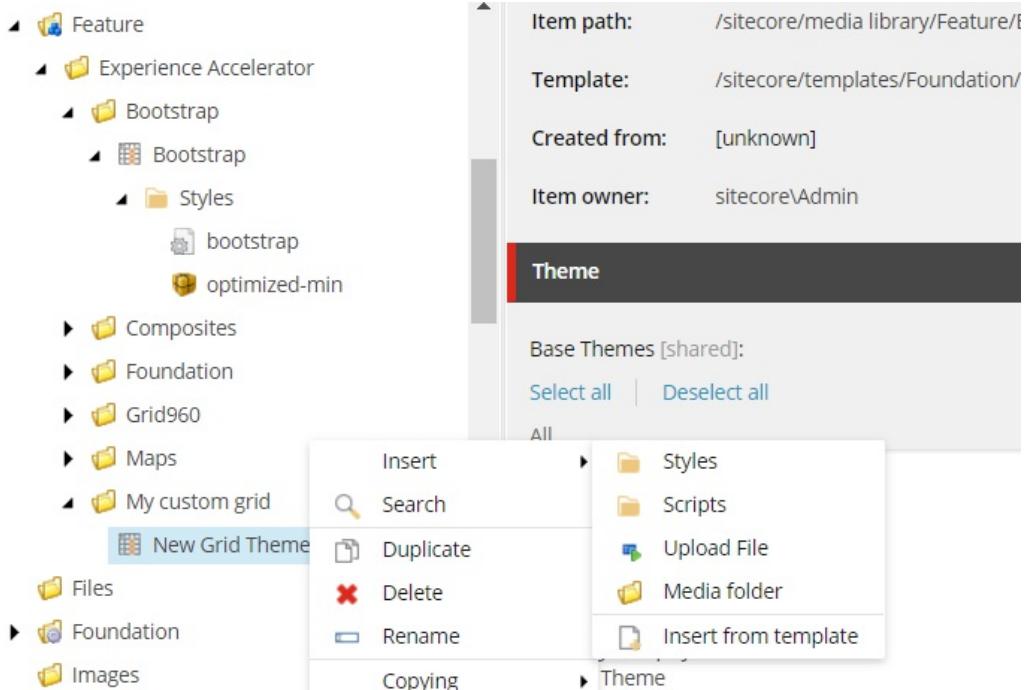
- [Create the Grid Definition item](#)
- [Create the grid scaffolding setup](#)

Create a grid theme

Your custom grid must have a grid theme assigned. SXA comes with a base grid theme that contains scripts and styles. You can use this base theme for your custom grid theme.

To create a grid theme:

1. Navigate to sitecore/Media Library/Feature and create a new folder for your grid. The default grids are saved in the Experience Accelerator folder (sitecore/Media Library/Feature/Experience Accelerator). Because SXA overwrites the standard SXA sections with SXA updates, we recommend that you create a separate folder for your grids.
2. Right-click the new folder and click Insert from template.
3. In the Insert from template dialog box, navigate to sitecore/templates/Foundation/Experience Accelerator/Grid/ and click Grid Theme.
4. Right-click the new grid theme and click Insert, Styles. Upload all your styles here.

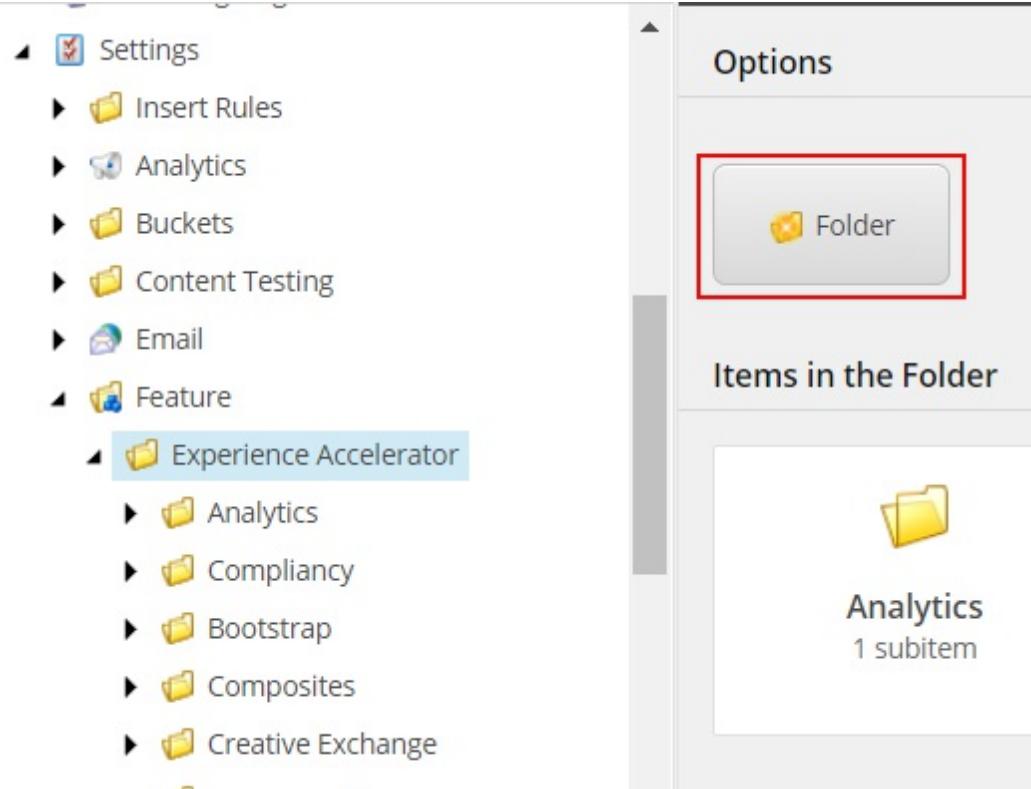


Create the grid definition item

Your custom theme needs a grid definition item that includes all grid-system specific items (the devices and their classes).

To create the grid definition item:

1. Navigate to sitecore/System/settings/Feature/ and create a new folder for your grid.



2. Right-click the new folder and click Insert from template.
3. In the Insert from template dialog box, navigate to /sitecore/templates/Foundation/Experience Accelerator/Grid/ and click Grid Definition.
4. Fill in the following fields:
 - Name - The name of the grid system as you want it to display in the site creation wizard.
 - Rendering parameters field type - Determines the way the grid parameters are rendered. If you do not want to customize, use the default field renderer:
`Sitecore.XA.Foundation.Grid.Fields.FieldRenderers.GridRen`
`Sitecore.XA.Foundation.Grid`
 - Default component layout - Sets the default column size for renderings that you drag on the page.

- Grid theme - Links to the theme used for the grid.
- Grid field parser type - Determines the parser type to parse the grid fields. If you do not need customization, use the default: `Sitecore.XA.Foundation.Grid.Parser.GridFieldParser`, `Sitecore`
- Grid body view path - Path to your cshtml file that defines the body layout of a page. You must include all necessary placeholders such as header, main, and footer.

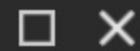
Create the grid scaffolding setup

To add the new grid as one of the grid options in the site creation wizard, you must add the grid setup.

To add the grid setup:

1. Right-click the new grid and click Insert from template.
2. In the Insert from template dialog box, navigate to `sitecore/Templates/Foundation/Experience Accelerator/Scaffolding/Roots` and click Grid Setup.
3. Fill in the following fields:
 - Name - The name of the grid system as you want it to display in the site creation wizard.
 - Dependencies - Specify the order to install the modules.
 - Grid Definition - Refers to the grid definition item. For example, for the custom grid in this example: `Settings/Feature/My custom grid/My custom grid definition`
4. You can now select your custom grid when creating a site.

Create a new SXA site.



This script will create a new fully functional site within your SXA instance.

GENERAL

MODULES

THEME

GRID

Grid

Bootstrap 4

Bootstrap 4

Bootstrap 3

Foundation

Grid 960

My custom grid

Designing

Abstract

This section describes how to work with page designs, partial designs, and themes.

To make structuring pages easier, SXA provides a mechanism for locking down designs and reusing them: Page Designs and Partial Designs. Read the topics in this section to learn more about page designs and partial designs.

- [Page designs](#)
- [Create and assign a page design in the Content Editor](#)
- [Create and assign a page design in the Experience Editor](#)
- [Create and change a partial design](#)

Themes define the look and feel of a site and can be created separately from the site functionality and content. Learn how to work with themes by reading the following topics:

- [The SXA themes](#)
- [Add a theme](#)
- [Assign a theme](#)
- [Extend a theme](#)

The SXA themes

Abstract

Learn more about the structure of base and site themes.

Themes define the look and feel of a site and can be created separately from the site functionality and content. There are two types of themes: base themes and site themes.

SXA comes with the default site theme named Wireframe to help you set up your site quickly. A site can be put together using the wireframe theme, while in the meantime the base theme is sent to a creative agency using Creative Exchange. When imported back, the site can be re-skinned using the new theme. You can [create a new theme](#) by copying the base theme and adding your own classes, applying a style to a particular rendering on a particular page, and adding assets, such as images, fonts, and files.

This topic describes:

- [Base themes](#)
- [Site themes](#)
- [The Wireframe theme](#)
- [The Basic2 theme](#)

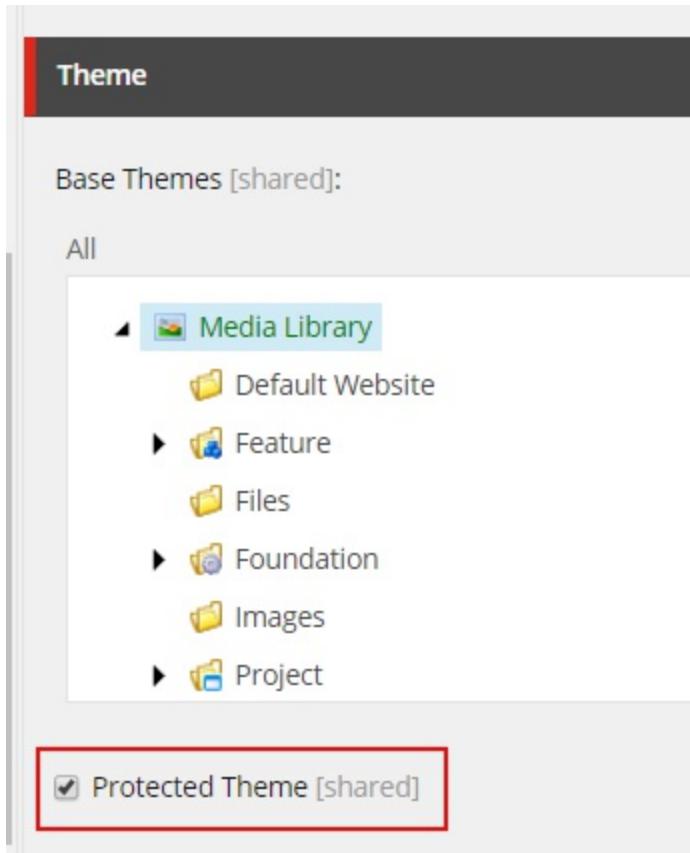
[Base themes](#)

Base themes are prototype themes that predetermine the layout of a website. You can have several base themes to support different design philosophies or specific functionality.

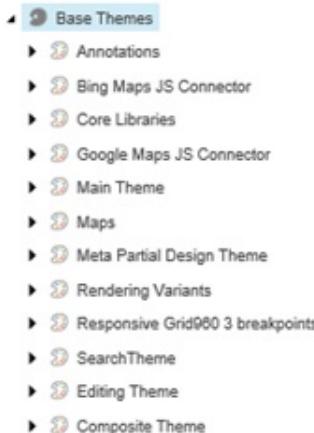
Note

Do not change base themes because these are part of the platform. If the base themes do not suit your needs, it is better to create a new base theme to inherit from.

Keep the Protected Theme checkbox selected for base themes. If the Protected Theme check box is selected, the theme is added to the `<restrictedLocations>` setting in `Sitecore.XA.Feature.CreativeExchange.config`, and changes to CSS/JS are not reflected when imported back using Creative Exchange.



The base themes are saved in the Media Library: /sitecore/Media Library/Base Themes.



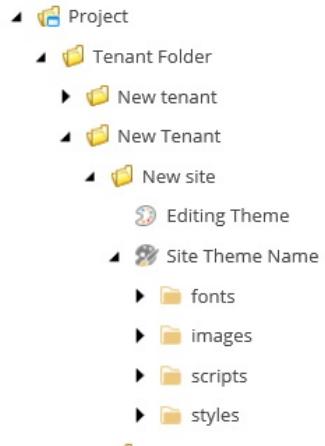
The Base Themes folder contains for example:

- Core Libraries – the third-party libraries used in projects such as: jquery, jquery UI, lo-dash, mediaelement, modernizr, and so on.
- Main Theme – the scripts and styles that are part of the platform (except for the rendering scripts). Main Theme has a dependency on Core Libraries, so if you are inheriting from it, you must also inherit from Core Libraries first.
- Grid themes – grid CSS generated by a sass grid generator. Determines the styling and logic for the chosen grid (for example, Bootstrap or Foundation)
- Composite theme - theme for renderings in composite view.
- Editing theme - logic used to edit components in Experience Editor.

Site themes

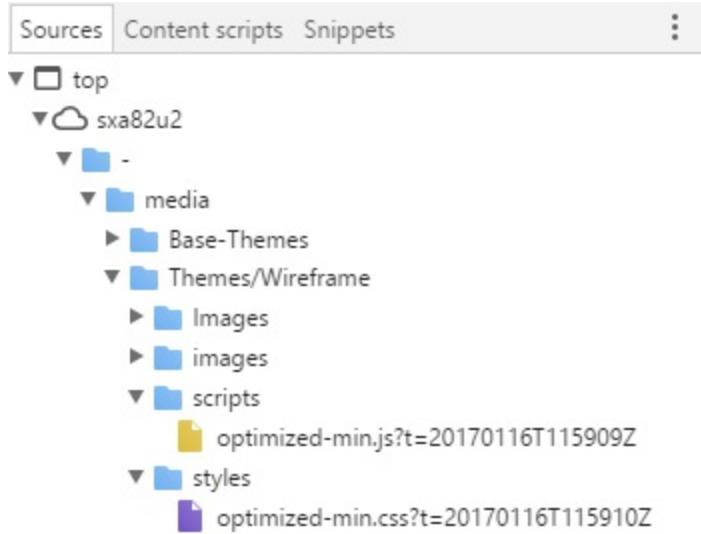
Site themes are extensions of base themes and can be applied to specific sites. Site themes usually have dependency on base themes and contain scripts and styles for all renderings used in a site.

A theme is a set of assets (style sheets, scripts, and images) that can be selected on a per-site basis. By default, a theme contains folders for fonts, images, scripts, and styles:

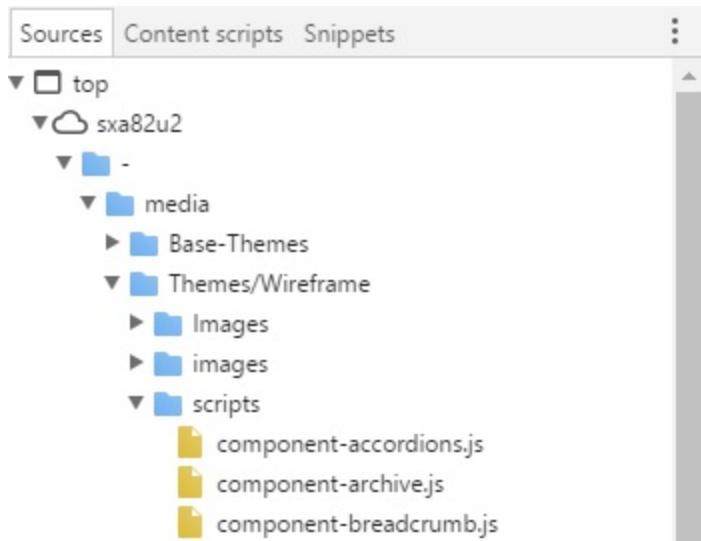


Folder Required	Description
fonts Not required	An example of how you could organize your fonts assets. This folder is optional and can be named differently.
images Required	An example of how you could organize your images assets. Although not directly required by SXA, some CSS may use image references to a set path.
scripts Required	Use to add your scripts or to delete unused scripts. Elements nearer to the top are loaded faster.
styles Required	Use to add or delete styles.

The Asset Optimizer uses the scripts and styles folders of the site theme to prepare minified source code (`optimized.min.js` and `optimized.min.css`) to save bandwidth during a page request:



If you have disabled the Asset Optimizer, instead of `optimized.min` files, you will see uncompressed scripts from the Media Library:



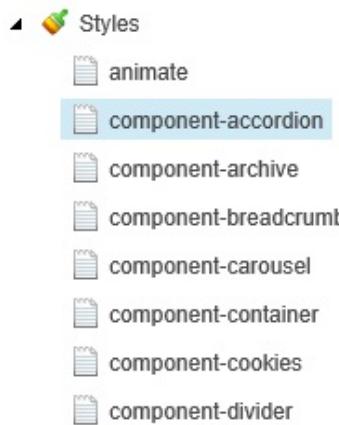
Note

You must not add folders under the `styles` and `scripts` folders. You can place your scripts and styles directly in the folders, without nested structure.

Site themes contain CSS styles, theme images, and JavaScript libraries used to provide your site branding. Site themes often contain Sass sources, compass config, used to generate CSS styles. In this case, front-end

developers should not modify the CSS files directly but rather work within the Sass workflow.

The CSS and JavaScript files in themes are divided into chunks that deal with one rendering at a time.



For each site theme, you must select a base theme to define the basic characteristics and properties of the theme. Every theme also needs a grid theme. SXA themes support multiple inheritance. This means that you can define multiple parent themes for every single theme. For example, in the following, the Holiday theme inherits from several other themes:

► Base Themes

◀ Themes

► Basic

◀ Holiday theme

► Fonts

► Images

► Scripts

► Styles

► New Branding

► Wireframe

Theme

Theme Base [shared]:

Select all | Deselect all

All

- Annotations
- Basic
- Bing Maps JS Connector
- Composite Theme
- Editing Theme
- Meta Partial Design Theme
- New Branding
- Rendering Variants
- Wireframe

Selected

- Core Libraries
- Main Theme
- Responsive Grid960 3 breakpoints
- Google Maps JS Connector
- Maps
- SearchTheme
- Holiday theme

The order of selected base themes is important. Be careful when you add a new theme or change the order. For example, if the SearchTheme relies on jquery library that is located in Core Libraries, you must make sure that the SearchTheme is loaded after the other themes.

The Wireframe theme

The standard SXA Wireframe theme that comes with core CSS and JavaScript is always the starting point for a new site. You can swap the Wireframe theme with a site theme once it has been prepared by your front-end developers.

Note

The Wireframe (deprecated) theme is deprecated from SXA version 1.8.

If you want to use mock images for the image placeholders, go to the Wireframe Image section and enter the fields:

- Mock Image - specify the image that displays as a mock image. If the field is left empty, no image will display.
- Stretch Image - when selected it stretches the image to full width/height.
- Background Color - specify the background color for the image in HTML hex format. If the field is left empty, the background will be transparent.

Wireframe Image

Mock Image [shared]:

Browse | Properties | Open media library | Edit image | Clear | Refresh

/Themes/Wireframe/images/imageicon



Dimensions: 56 x 40
Alternate Text: "Mock image for Wireframe theme"

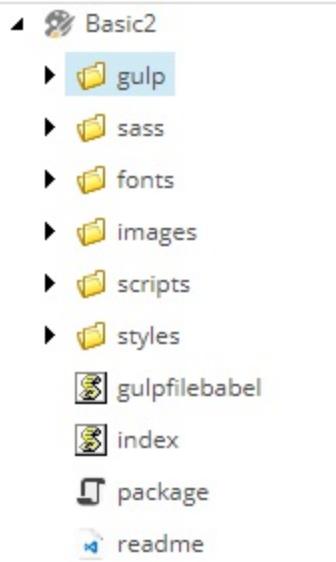
Stretch Image [shared]

Background Color [shared]:

#E3E3E3

The Basic2 theme

You need the Basic2 theme to be able to work with [Creative Exchange Live](#). Creative Exchange Live is an SXA feature that enables front-end developers to modify themes and other site content without having to import the files back into the site. The Basic2 theme allows you to work with gulp.



	Folder Required	Description
gulp	Required	To use Creative Exchange Live, your site must use the Basic2 theme. This theme contains the Gulp folder. Gulp tasks that enable you to make changes to themes and other content and synchronize to the Sitecore environment immediately.
sass	Required	Folder that contains all Sass files: scripts and styles.
fonts	Not required	An example of how you could organize your fonts assets. This folder is optional and can be named differently.
images	Required	An example of how you could organize your images assets. Although not directly required by SXA, some CSS

may use image references to a set path.

scripts Required Use to add your scripts or to delete unused scripts.
Elements nearer to the top are loaded faster.

styles Required All CSS styles for renderings that will be created as a result of compiling Sass files.

Assign a theme

Abstract

Change the site theme with Content Editor or Experience Editor.

In SXA, you can style your pages using themes. Themes let you change the style of an existing site, for example, because of company branding changes or if you want a holiday edition, without interfering with the content. You can use Creative Exchange to export and import themes.

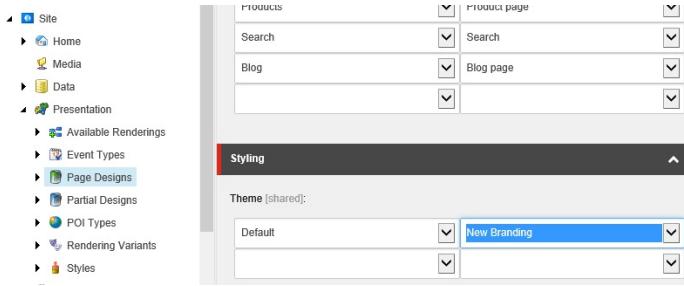
You can assign a different theme in the Experience Editor or in the Content Editor.

To assign a theme:

- In the Experience Editor, on the ribbon, on the Experience Accelerator tab, click Theme, and in the Select Theme dialog box, select the theme you want to use. Click OK to save your changes.



- In the Content Editor, on your site, click Presentation, Designs, and in the Styling section, select the theme you want to use.



Note

If you do not select a theme, the default theme is assigned.

Extend a theme

Abstract

Learn how to create a theme extension and use the Attach Theme Extension or the Extend Site Theme actions.

SXA uses themes to define the look and feel of a site. Themes can be created separately from the site functionality and content. You can extend an existing theme without having to change the theme itself. For example, to enable or disable certain theme features based on selected site features, to extend SXA themes such as the Basic theme and the Wireframe theme, or to extend existing themes without having to create a completely new theme or to add styles, images, and/or JavaScript after every SXA update.

You extend a theme by creating a theme extension and using the Attach Theme Extension or the Extend Site Theme action.

This topic describes how to:

- [Create an extension for a theme](#)
- [Extend a theme using the Extend Site Theme action](#)
- [Extend a theme using the Attach Theme Extension action](#)

Create an extension for a theme

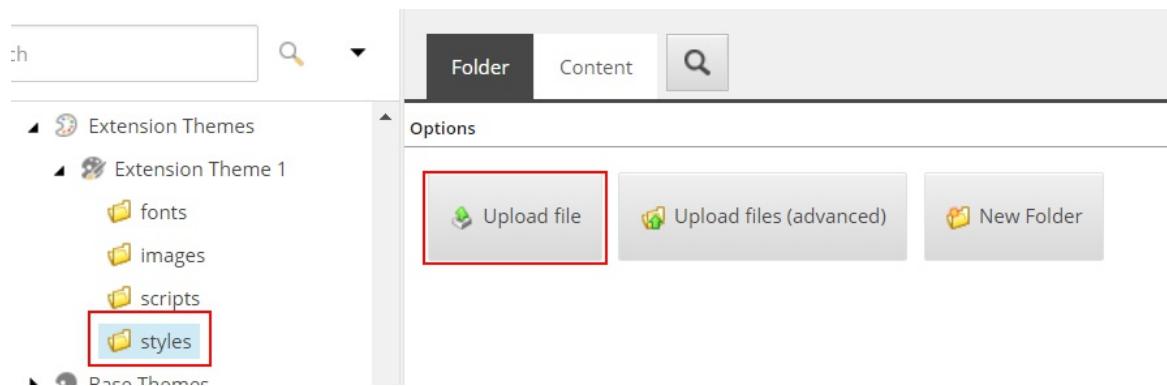
Creating theme extensions instead of changing a theme gives you the flexibility to use the extensions where you need them. For example, if you want different styles to be available for footers and headers.

To create the extension of a theme:

1. In the content tree, navigate to the Media Library and right-click

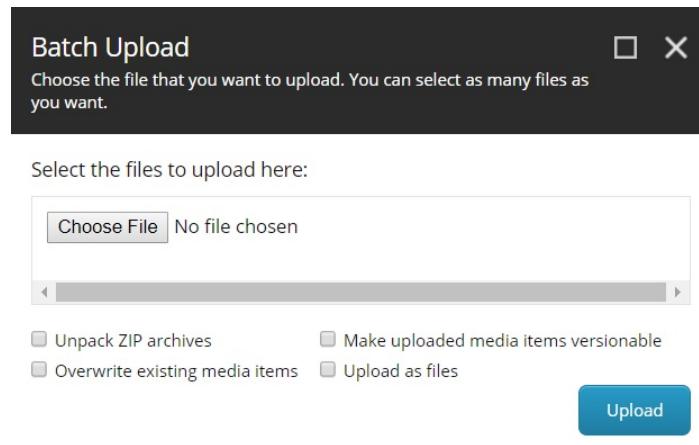
Extension Themes, click Insert and click Extension Theme.

2. Enter a name, and click OK.
3. Now you can upload styles and media items. For example, to upload a CSS file that changes the styling of the footer, click the styles folder and click Upload File.



Note

To upload many different files, click Upload files (advanced) and batch upload styles and items.



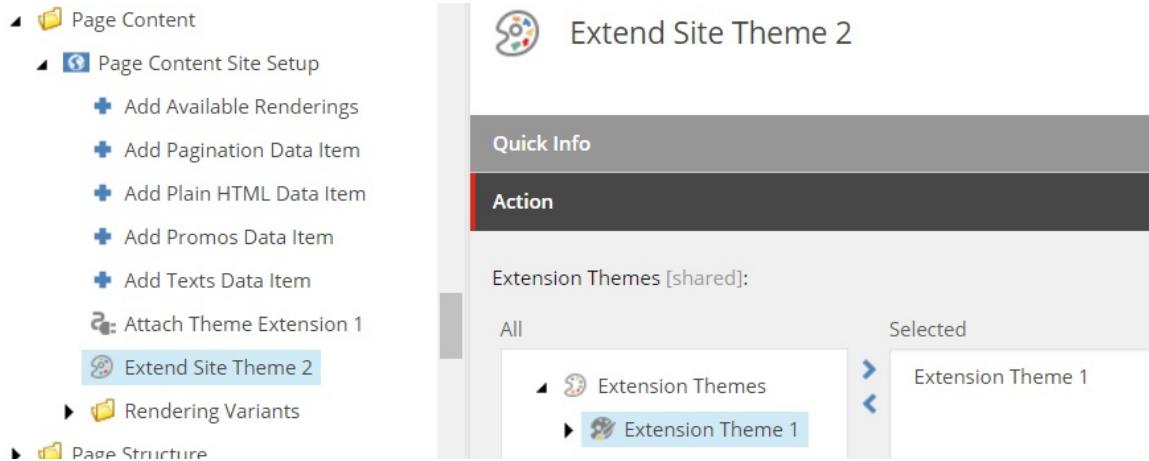
4. In the Upload File dialog box, click Choose File, select the style you want to add and click OK.

Extend a theme using the Extend Site Theme action

To use your theme extension for new themes, you must use the Extend Site Theme action. For example, when you create a new site and with that site you create a new site theme, the theme extension is automatically incorporated for the features that have theme extensions selected.

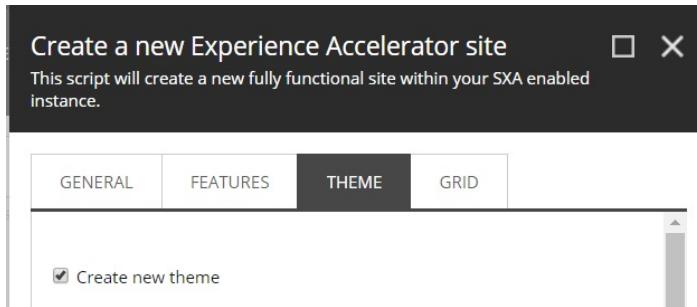
To extend a theme by using the Extend Site Theme action:

1. Navigate to System\Settings\Feature\Experience Accelerator and click the Site Setup folder of the feature that you want to extend. For example, the Page Content feature. Right-click Page Content Site Setup, click Insert, and click Extend Site Theme.
2. In the Extension section, in the Extension Themes field, move the Extension theme to the selected field.

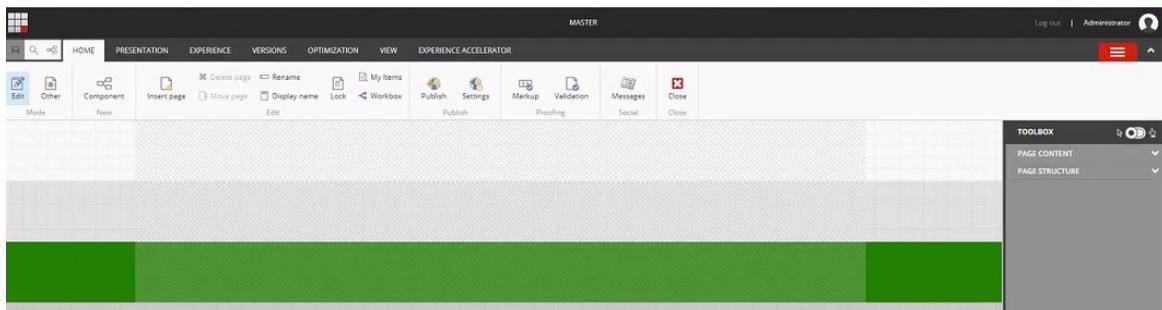


3. Save the changes.

Now, when you create a new site with the Page Content feature enabled and with the Create new theme check box selected, your theme will be extended.



In this example, we added a new footer style to the Page Content feature. If you create a page for the new site and open that page in Experience Editor, the updated footer displays.



Note

To use the extended theme in existing sites, you must assign the theme to the site pages.

[Extend a theme using the Attach Theme Extension action](#)

To deploy a theme extension separately, you must use the Attach Theme Extension action.

To add a theme extension using the Attach Theme Extension action:

1. Navigate to System\Settings\Feature\Experience Accelerator and click the rendering you want to add a style for. For example, to add the new footer style, right-click Page Content Site Setup, click Insert, and

click AttachThemeExtension.

2. In the Extension section, in the Theme that is extended field, move the theme that you want the extension to apply for to the selected field. For example, the Wireframe theme.

The screenshot shows the Sitecore 'Page Content' tree on the left and the 'Extension' dialog on the right. The tree includes items like 'Page Content', 'Page Content Site Setup' (with options for renderings, pagination, plain HTML, promos, texts, and attach theme extension), 'Rendering Variants', 'Page Structure', 'Redirects', and 'Search'. The 'Extension' dialog has 'Item owner: sitecore\Admin'. The 'Extension' tab is selected. Under 'Theme that is extended [shared]:', there is a list of themes: 'Themes' (Basic (Deprecated), Basic2, Showcase, Wireframe). The 'Wireframe' theme is highlighted with a red box and labeled 'Selected'.

3. In the Extension section, in the Extension Themes field, you can now move the Extension theme to the selected field.

The screenshot shows the Sitecore 'Page Content' tree on the left and the 'Extension' dialog on the right. The tree includes items like 'Page Content', 'Page Content Site Setup' (with options for renderings, pagination, plain HTML, promos, texts, and attach theme extension), 'Rendering Variants'. The 'Extension' dialog has 'Item owner: sitecore\Admin'. The 'Extension Themes [shared]:' section shows a list of extension themes: 'Extension Themes' (Extension Theme 1). 'Extension Theme 1' is highlighted with a blue box and labeled 'Selected'.

4. Save the changes.

When you open a page in Experience Editor, the updated footer displays.

Create a custom theme

Abstract

Define the look and feel of your site in a theme.

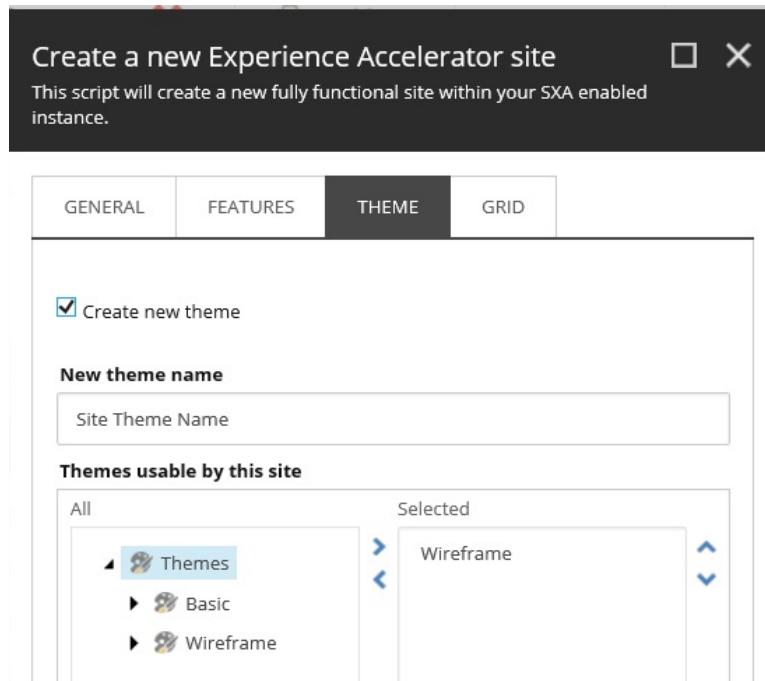
Themes define the look and feel of a site and can be created separately from the site functionality and content. By default, SXA comes with two types of themes:

- Site Themes (/sitecore/Media Library/Themes) – branding themes that contain scripts and styles that are responsible for the look and feel of your site. For example, renderings styling (navigation, carousel, and so on) and renderings behavior (if necessary a custom one).
- Base Themes (/sitecore/Media Library/Base Themes) – foundation themes that contain scripts and styles that deliver more complicated functionalities. For example, shared functionalities such as Bing/Google maps connector, Core Libraries (jquery, lo-dash), and scripts that influence the editing experience (sticky notes, editing themes, drag and drop, and partial designs highlighting).

To add your own classes and assets, such as styles, scripts, images, and fonts, you can create a new site theme.

To create a custom theme:

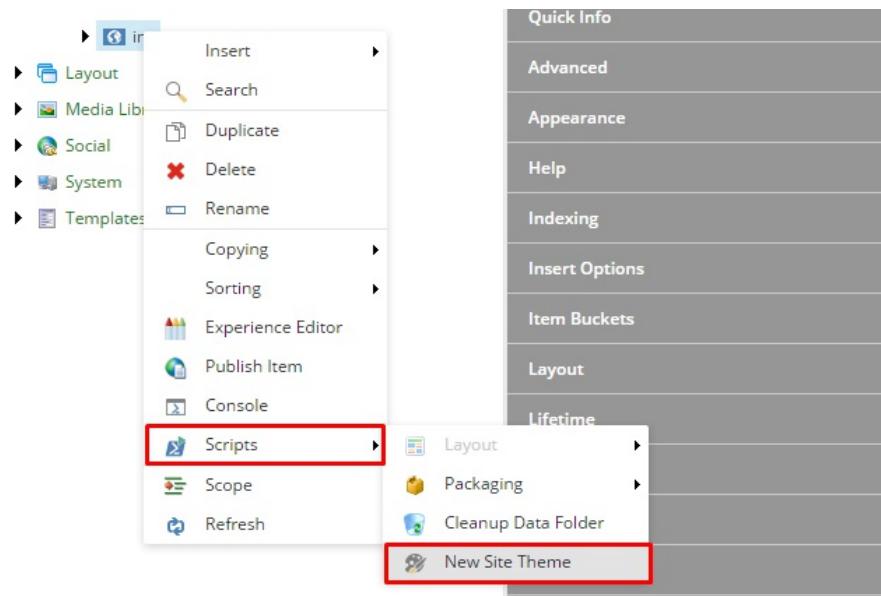
1. Add a new theme in one of the following ways:
 - Add a theme when creating your site. In the Create a new Experience Accelerator site dialog box, on the Theme tab, select the Create new theme checkbox.



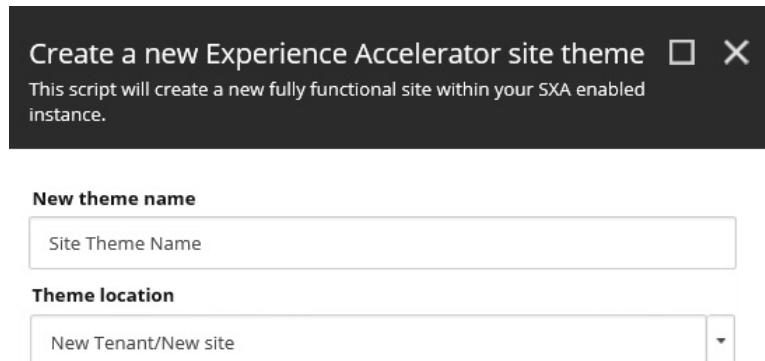
The new theme is added to the Site folder in the Media Library:

/sitecore/Media Library/Project/Tenant Folder/Your Tenant/Your Site/Site Theme Name

- Add a theme manually. Add a theme under your tenant/site media library folder. Copy all children of /sitecore/media library/Themes/Basic theme.
- Add a theme using a script. SXA contains a helper script that creates a new theme for you. Right-click your site, click Scripts, and click New Site Theme.



In the Create a new Experience Accelerator site theme dialog box, enter a name and optionally enter a new location for your theme. By default, the new theme is added to: /sitecore/Media Library/Project/Tenant Folder/Your Tenant/Your Site/Site Theme Name.

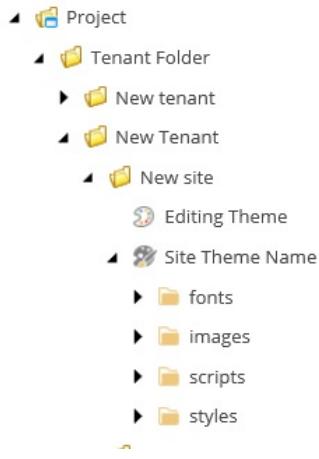


Note

By default, new themes are stored here: /sitecore/Media Library/Project/Tenant Folder/Your Tenant/Your Site/Site Theme Name. Do not save your theme items under any of the SXA roots as they might be overwritten in the next release.

2. By default, the new theme contains the following folders: fonts, images,

scripts, and styles. Use the scripts and styles folder to add CSS styles and JavaScripts.



Note

The fonts folder is optional and is an example of how you could organize your fonts assets. The image folder is also optional but some CSS can use image references to a set path.

3. In the Internet Explorer Compatibility section, set the X-UA-Compatible value. This value forces Internet Explorer to use a specific Edit and Preview/Public mode to render your web pages. For example, you can set the value to IE Edge to use the highest, most recent mode available.

The screenshot shows a configuration window titled "Internet Explorer Compatibility Mode". It contains two sections: "Compatibility in Edit Mode [shared]" and "Compatibility in Preview & Public Mode [shared]". The "Compatibility in Preview & Public Mode" section is expanded, showing a list of browser versions: Edge, Internet Explorer 11, Internet Explorer 10, Internet Explorer 9, Internet Explorer 8, and Internet Explorer 7. A dropdown arrow is visible next to the list.

4. In the Theme section, set the themes your new theme should inherit from. SXA themes support multiple inheritance. This means that you can define multiple parent themes for every single theme.

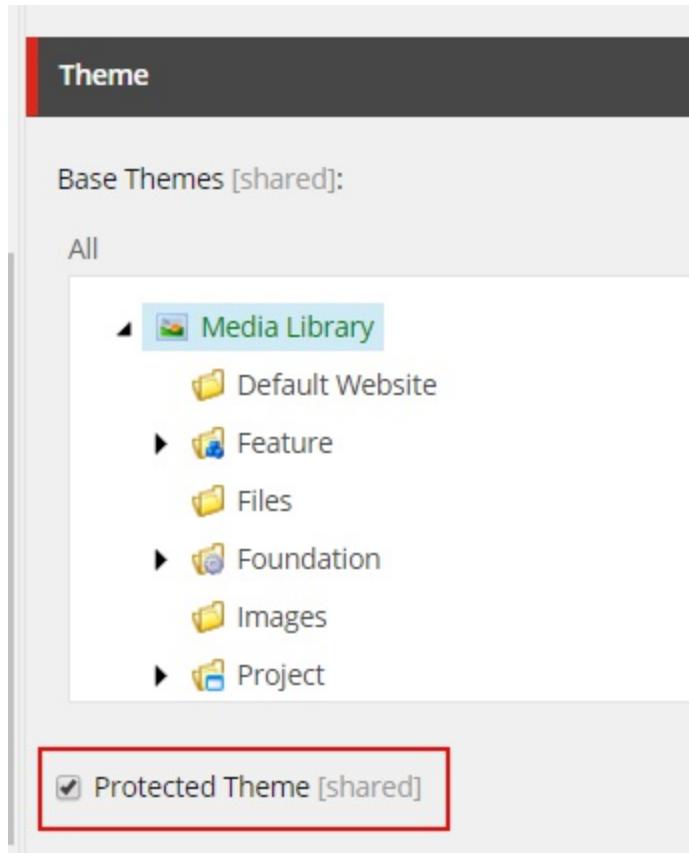
Note

The order of selected base themes is important. Be careful when you add something new or change the order. For example, if the SearchTheme relies on jquery library that is located in Core Libraries, you must make sure that the SearchTheme is loaded after the other themes.

The screenshot shows a list of themes under the heading "Core Libraries". The themes listed are: Main Theme, Grid960, Google Maps JS Connector, Maps, and SearchTheme. The "SearchTheme" item is highlighted with a blue background.

If you created a new base theme or just don't want your theme to be changed, select Protected Theme. This will add the theme to the list of

locations that are protected from modification and prevents Creative Exchange's import function from updating the theme
(in Sitecore.XA.Feature.CreativeExchange.config: experienceAcce



5. If you want your images to be turned into wireframe images, select the Support Wireframe Images check box.



6. In the Global Classes section, you can override some of the standards that come with the SXA grid and specify that your footer, header, and content placeholders will have full width.
 - WideHeader – select to make the header placeholder use 100% of the screen width.
 - WideContent – select to make the content placeholder use 100% of

the screen width.

- WideFooter – select to make the footer placeholder use 100% of the screen width.
 - OtherClasses – specify a value to render into a class attribute of the HTML body tag on each page using this theme.
7. In the Viewport field, enter a <meta> viewport element. The viewport is the user's visible area of a web page that varies with the device. You can take control over the viewport, using the <meta> tag. This <meta> viewport element gives the browser instructions on how to control the page dimensions and scaling. If your meta partial design contains a Viewport rendering, the value of this field is used to fill this meta tag. For example:



Renders the following on the page:

```
<meta name="viewport" content="viewport_value">
```

8. If you want to use mock images for the image placeholders, go to the Wireframe Image section and enter the fields:

Mock Image - specify the image that displays as a mock image. If the field is left empty, no image will display.

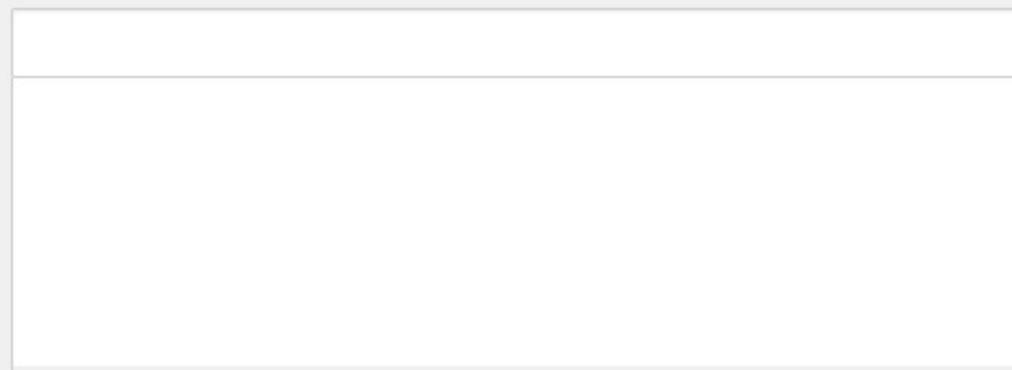
Stretch Image - when selected it stretches the image to full width/height.

Background Color - specify the background color for the image in HTML hex format. If the field is left empty, the background will be transparent.

Wireframe Image

Mock Image [shared]:

[Browse](#) | [Properties](#) | [Open media library](#) | [Edit image](#) | [Clear](#) | [Refresh](#)



This media item has no details.

Stretch Image [shared]

Background Color [shared]:

#e3e3e3

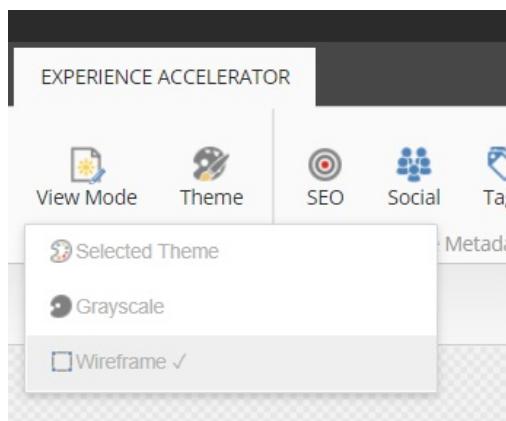
The View Modes

Abstract

Switch view modes

SXA comes with different view modes to help you set up your site quickly. A site can be put together using wireframe mode, while in the meantime the site theme is sent to a creative agency using Creative Exchange.

When you open a site that has no site theme selected in Experience Editor, the Wireframe mode is automatically selected. On the ribbon, on the Experience Accelerator tab, click View Mode to switch the mode.



You can select the following view modes:

- **Wireframe** - Wireframe theme is selected. Creative Exchange shows a warning during export.

f the most basic parts of every web page is images. Thanks to this component you can put graphics into your pages and customize them: change the default dimension, e alternative text and enter some captions underneath.

Image component

Drop the component on the page and select a picture from the Media Library. If this simple setup is not enough you can modify the width and height and provide alternative text, as with every Sitecore image . But with Sitecore Experience Accelerator you can make your image link to a URL or add a caption under the picture.

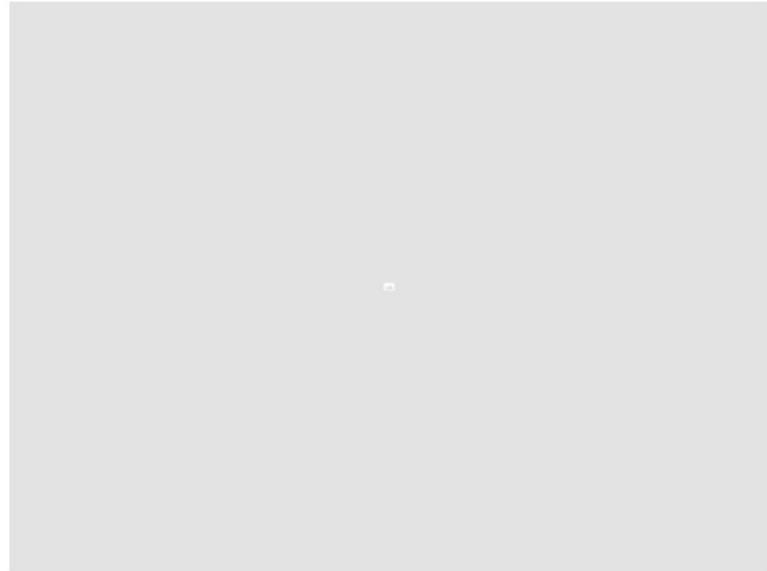
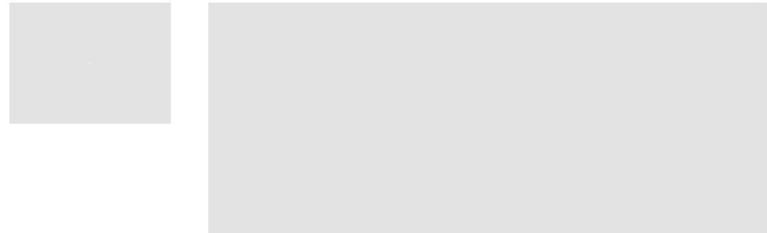


Image size

The component will automatically fit the image to the size of the containing placeholder or it will resize it to whatever dimensions you specify.



- **Grayscale** - The Site theme is selected and the styling is changed to remove all colors from styles and images. Creative Exchange shows a warning during export.

One of the most basic parts of every web page is images. Thanks to this component you can put graphics into your pages and customize them: change the default dimension, provide alternative text and enter some captions underneath.

Image component

Drop the component on the page and select a picture from the Media library. If this simple setup is not enough you can modify the width and height and provide alternative text, as with every Sitecore image. But with Sitecore Experience Accelerator you can make your image link to a URL or add a caption under the picture.



[No text in field]

Image size

The component will automatically fit the image to the size of the containing placeholder or it will resize it to whatever dimensions you specify.



[No text in field]



- Selected Theme - Site theme is selected.

SITECORE EXPERIENCE ACCELERATOR

DESIGNS Base Partial Design Site Page Designs

Look & Feel

View Mode Theme SEO Social Tags Other

Selected Theme ✓

Grayscale

Wireframe

Page Metadata

Creative Exchange Publish

Export Import

Go Edit

Picture from the Media

This simple setup is not

More ▾

You can modify the width and height and provide alternative text, as

every Sitecore Image . But with

Sitecore Experience Accelerator you

make your image link to a URL or

a caption under the picture.

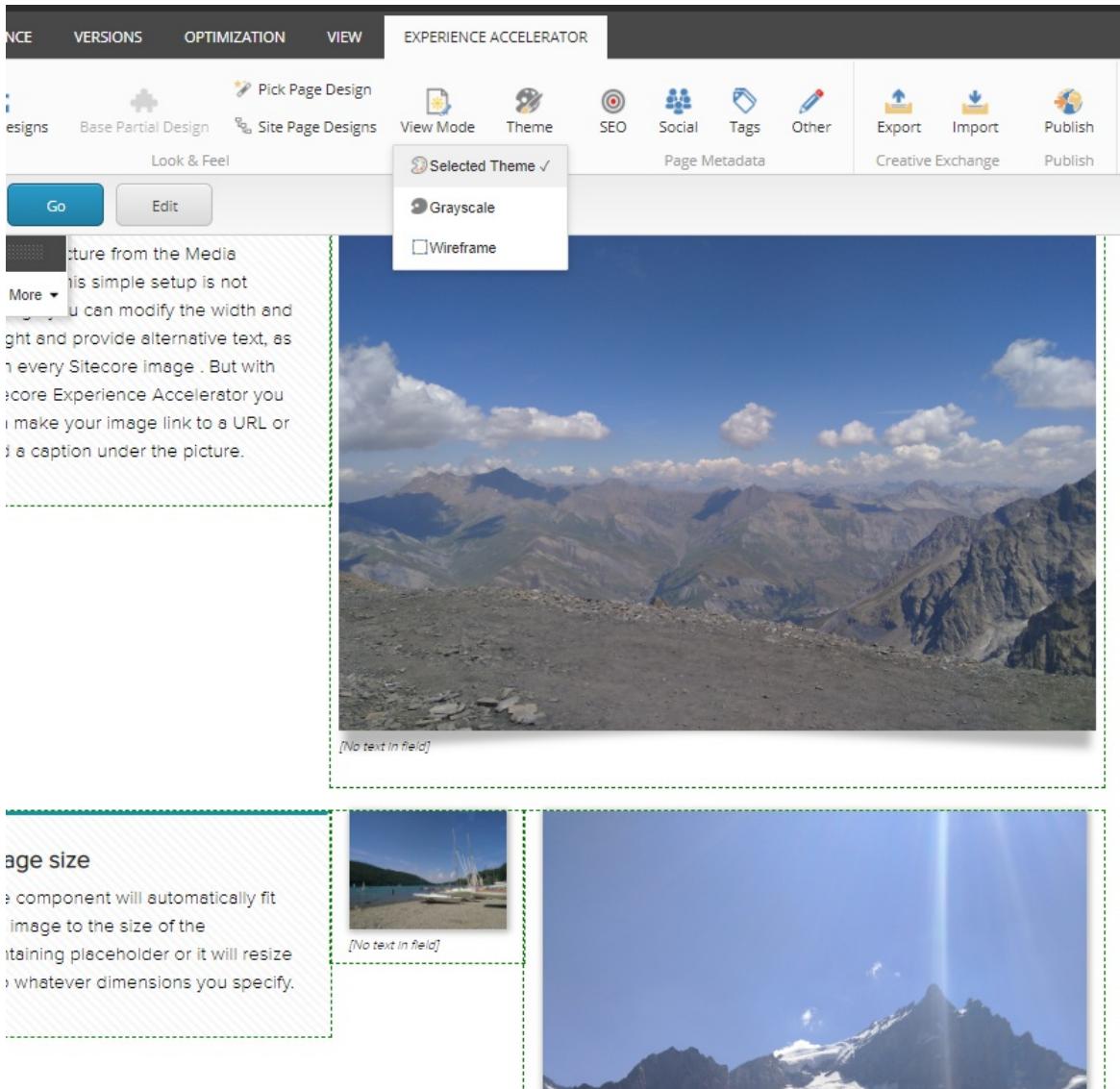
[No text in field]

Image size

This component will automatically fit the image to the size of the containing placeholder or it will resize it whatever dimensions you specify.

[No text in field]

[No text in field]



Page designs

Abstract

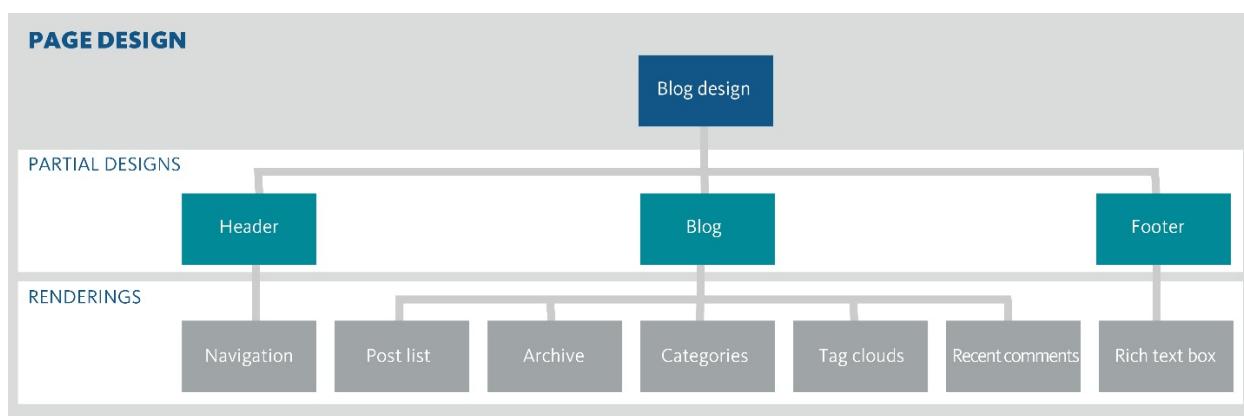
A page design in SXA is a group of partial layouts that make up the design of a webpage.

In SXA, you build your website with reusable pieces of content and layout. All these pieces together create the layout for your pages.

A page design in SXA is a selection of [partial designs](#) and renderings that help you to structure your pages. You can, for example, make sure the headers and footers are always in the same place. You can also create a page design set up a page structure for specific pages, such as a blog page, a landing page, a product page, and so on. Content authors can then place content in these preset layouts.

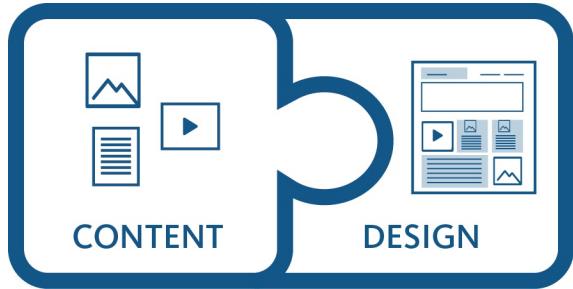
You can [create page designs in the Content Editor](#), and in the [Experience Editor](#).

You assign a page design to a page (or pages of a specific type) to define the elements and renderings that you want to appear. For example, a blog page may need a header with a navigation component, a main placeholder for the content (post list, categories, archive, tag cloud, recent comments), and a footer with company information.



Data templates are the schema for Sitecore content. Any content item in a Sitecore database is based on a data template.

In SXA, you can link page designs to data templates. In this way, you can link your content types to your page layouts and keep the layout of your site consistent. It is very convenient to assign a page design to the template for web pages that you know you will be using a lot, so that they look consistent.



Create and assign a page design in the Content Editor

Abstract

Work with groups of partial layouts to create the design for a page.

In SXA, you work with reusable pieces of content and layout. A page design in SXA is a group of partial layouts that make up the design of a webpage.

For example, you may need a reusable page structure for a blog page. By creating a page design that includes the partial designs and renderings that you need for a blog page, you create a basic template that content authors can use to create content without having to worry about the design of the page.

You can work with SXA page designs in both the Content Editor and the Experience Editor.

This topic outlines how to:

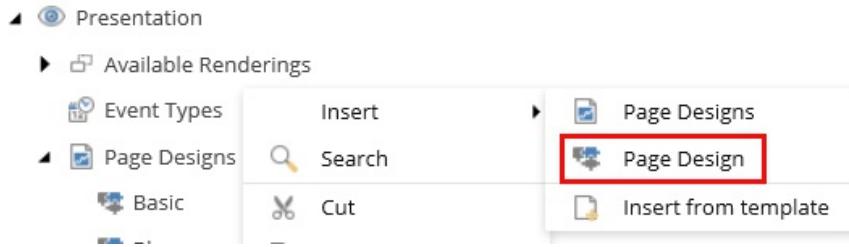
- [Create a Page Design](#)
- [Assign a Page Design](#)

Create a Page Design

A page design determines the layout of a page and consists of partial designs and renderings. You can add page designs to the presentation folder of your site and select the partial designs that you want to add.

To create a page design:

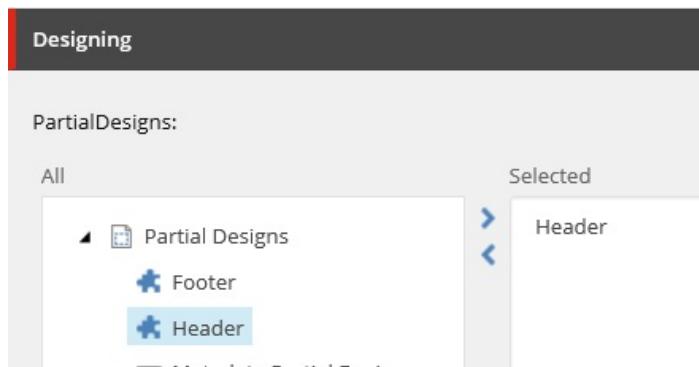
1. In the content tree, on your site, click Presentation, right-click Page Designs, click Insert, Page Design.



Note

If you want to add a group of related page designs, click Page Designs. This can be convenient if you have a complex site that requires a large number of partial designs. You can divide your page into sections of partial designs, for example, blogs, news, products, and careers.

2. Enter a name for the new page design and click OK.
3. In the Designing section, select the partial designs that you want to add, click the right arrow to move them to the list of selected items, and then save it.



4. You can also override the general theming of your site and assign a specific theme to a page design. In the Theming section, click the drop-down arrow and select the theme..
5. Right-click the new page design, and then click Experience Editor to view your design.

Note

If a page design is not in use, you may want to delete it. To delete a page design in the Content Editor, right-click the page design and click Delete.

Assign a Page Design

You can use Page Designs to map content types to your page layouts. By doing this, you keep the layout of your site consistent. For web pages that you use often, such as landing pages, product pages, and navigation pages, you can keep them consistent by assigning a page design to the template for that page. In this way, you link the content to the design.

To assign a page design to a template:

1. In the content tree, go to your site, click Presentation, Page Designs.
2. In the Designing section, select a template in the left column and, to associate it to a page design, in the right column, select the page design.

Note

You can also assign page designs to specific pages. This may be necessary when you need a page that you normally don't use very often, such as for example a release notes page. To assign a page design to a single page, go to your site and select the page. In the Designing section, select the design from the drop-down list and save the changes.

Right-click the page and click Experience Editor to view the result.

Create and assign a page design in the Experience Editor

Abstract

Work with groups of partial layouts to create the design for a page.

In SXA, you work with reusable pieces of content and layout. A page design in SXA is a group of partial layouts that make up the design of a webpage.

For example, you may need a reusable page structure for a blog page. By creating a page design for a blog, you can create a basic template that content authors can use to create content without having to worry about the design of the page.

This topic outlines how to:

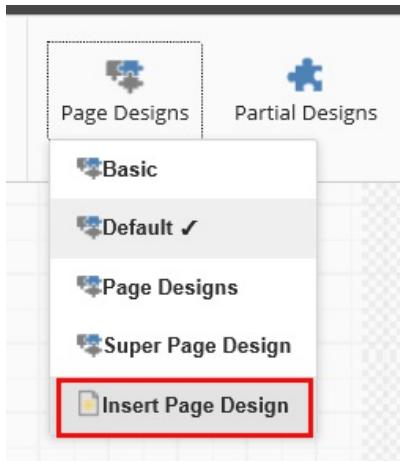
- [Create a page design](#)
- [Assign a page design](#)

Create a page design

In SXA, you can set up a page design to determine the layout of a page.

To create a page design in the Experience Editor:

1. On the ribbon, on the Experience Accelerator tab, click Page Designs, and then click Insert Page Design.



2. In the Insert Item dialog box, click Page Design, enter a name, and click OK.

Note

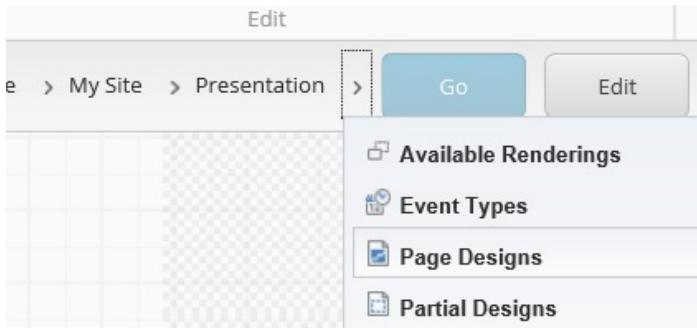
If you want to add a group of related page designs, click Page designs. This can be convenient if you have a complex site that requires a large number of partial designs. You can divide your page into sections of partial designs, for example, blogs, news, products, and careers.

3. In the Select items dialog, select the Partial Design(s) that you want to add, click the right arrow ➤ to move it to the list of selected items, and click OK.

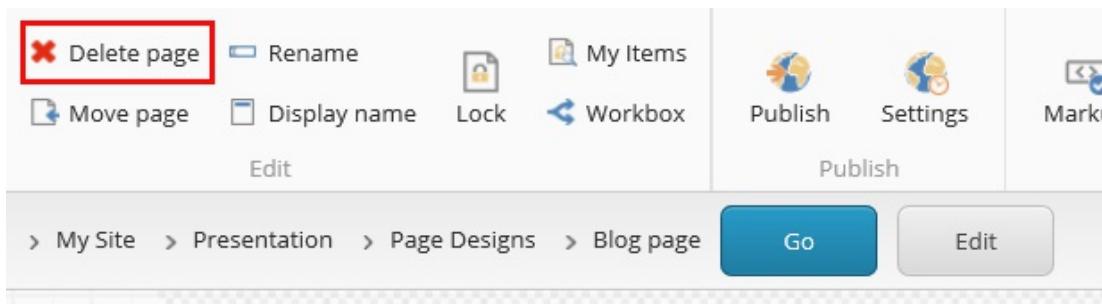
When you are no longer using a page design, you may want to delete it.

To delete a page design in the Experience Editor:

1. On the ribbon, on the View tab, select Navigation bar, go to the Presentation layer of your site and click the page design that you want to delete, and then click Go.



2. On the Home tab of the page design, click Delete page.

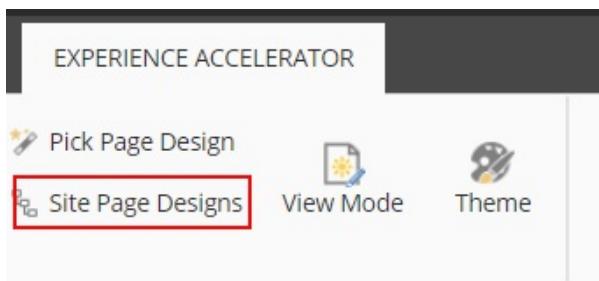


Assign a page design

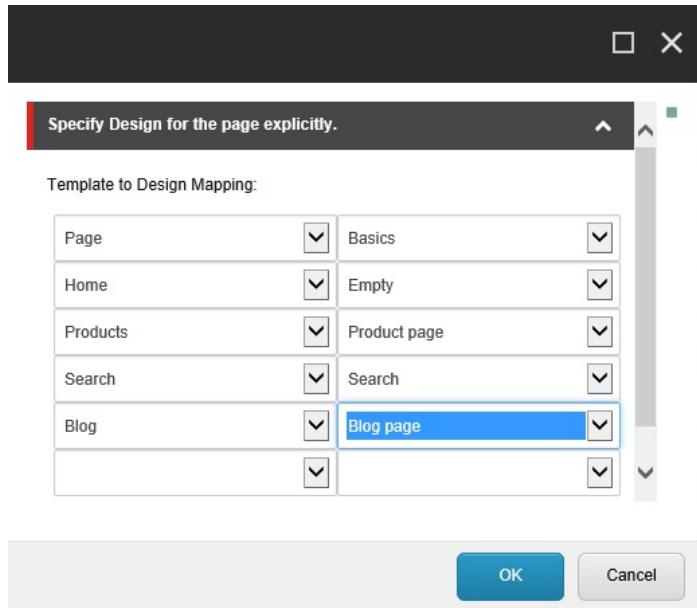
You can use page designs to map content types to your page layouts. By doing this, you keep the layout of your site consistent. For web pages that you use often, such as landing pages, product pages, and navigation pages, you can keep them consistent by assigning a page design to the template for that page. In this way, you link the content to the design.

To assign a page design to a template:

1. On the ribbon, on the Experience Accelerator tab, click Site Page Designs.

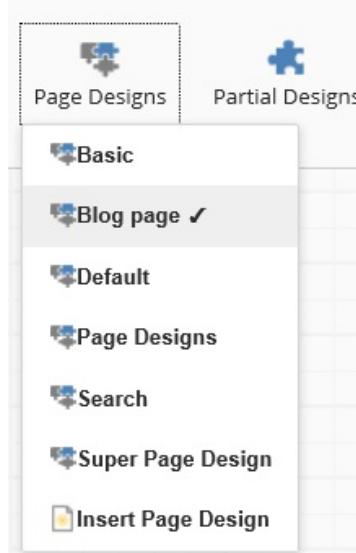


2. In the dialog box that appears, in the left column, select a template and, to associate it to the page design, in the right column, select the page design.



3. Click OK and Save to apply the changes.

To see the page design assigned to a page, click Page Designs. The assigned page design is marked, for example, Blog page.



Note

If you don't use a particular design very often or you want to override a particular page design, you can assign a page design to a specific page. To do this, on the ribbon, on the Experience Accelerator tab click Pick Page Design, select the page, click OK and Save to apply the changes.

Create and change a partial design

Abstract

Learn how to create a partial layout in the Content Editor or Experience Editor.

SXA uses sets of renderings, called partial designs that make up a webpage. You can use the partial designs to create the design elements of your pages quickly for a consistent style. For example, you can create parts of your page once, such as headers and footers, and then use them everywhere on your site.

You can also change a partial design for a specific page, for example, if you need a slightly different header on a particular page. Partial designs can inherit from each other, so you can build increasingly complex designs from a basic set of reusable partial designs.

This topic outlines how to:

- [Create a partial design](#)
- [Change a partial design](#)

[**Create a partial design**](#)

If the existing partial designs do not suit your needs, you can create a new partial design to reuse across pages, for example, a partial design for a page header.

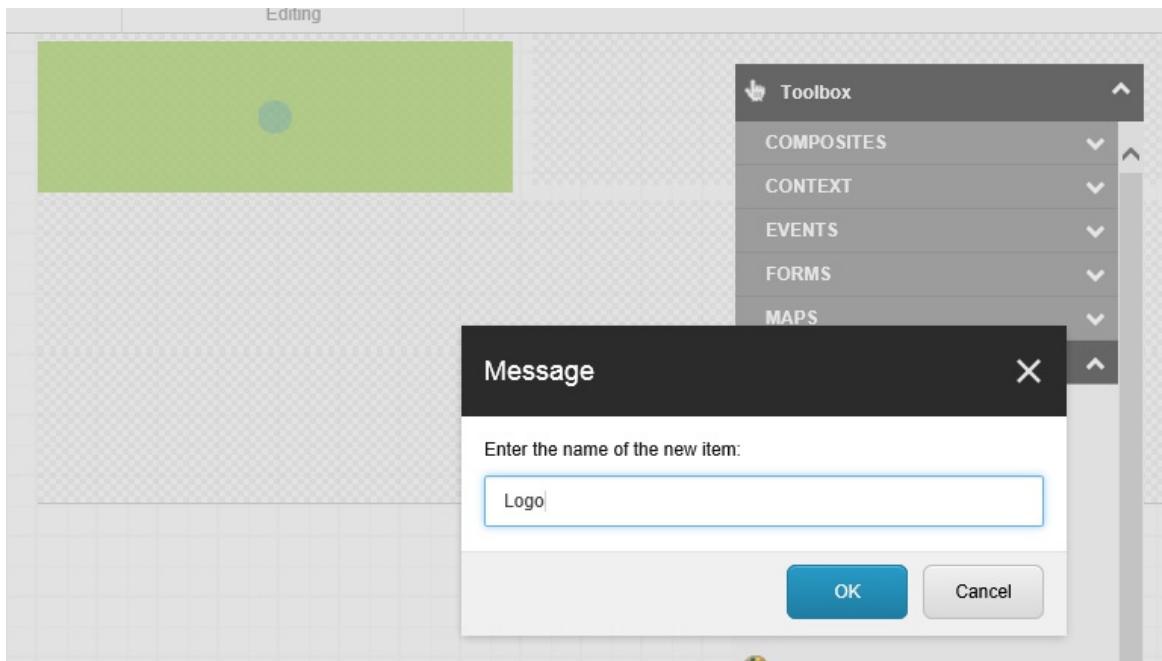
To add a partial design in the Experience Editor:

1. On the ribbon, on the Experience Accelerator tab, click Partial Design, and then click Insert a new Partial Design.
2. In the Insert Item dialog box, select Partial Design, enter a name and

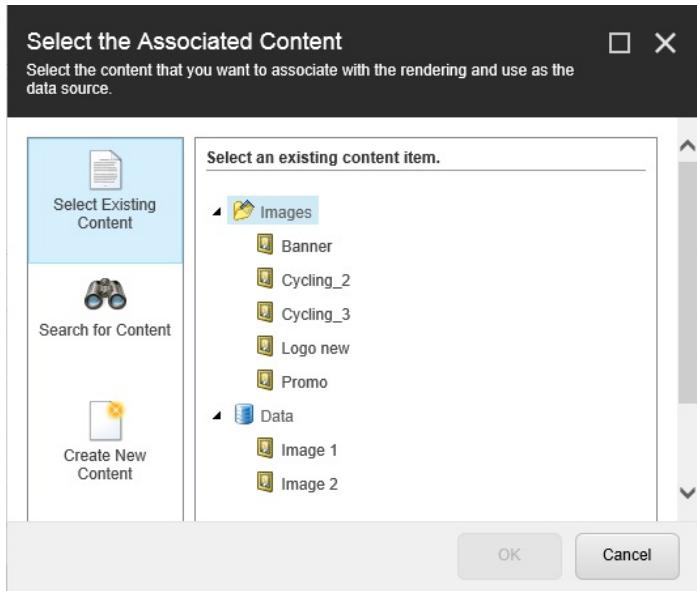
click OK. SXA now loads an empty page. If you select the navigation check box, you can see you are in the presentation layer.



3. Now you can add renderings to your partial design by dragging them from the Toolbox to the page. For example, for a header, you could add some layout elements and divide the header in 4 columns on the left and 8 on the right. Drag the Splitter (Columns) rendering to the page and split the layout.
4. Use the Image (Reusable) rendering to add a logo to the left 4 columns. To do that, drag the Image (Reusable) rendering to the left columns, enter a name and click OK.



5. In the Select the Associated Content dialog, select the image and click OK.



Note

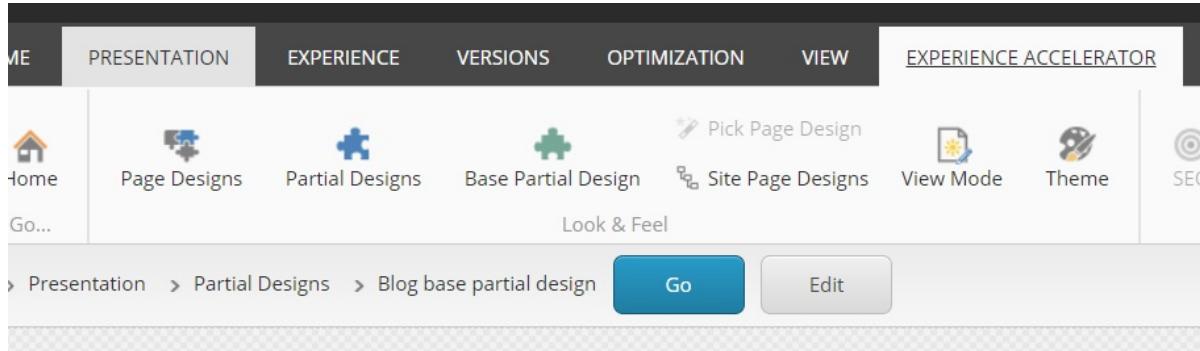
If you select a data source under the Page Data node it will be a local data source that is stored as a subitem of the page item. Any changes you make to local data sources will only affect the page you are working on. If you want to be able to reuse the data source and manage it globally, select a different place in the tree.

When your header is ready after for example adding a slogan to the header, and adding navigation, save it to make it available in the Partial Design menu.

Create a base partial design

Two slightly different headers. There is no need to create a number of redundant Partial Designs. By taking advantage of Partials inheritance, you can create one Base Partial Design with - for example - Navigation Component, and then create other Partials on the top of that one and fill in the differences with different logos, etc. This way you avoid the creation of numerous templates with only minor differences. This has been a major

problem for CMS systems. With Partials inheritance, you can maintain the shared aspects of your design in one place and easily propagate those changes to all the different page types that the authors are creating.



Change a partial design

Occasionally, you may want to change a partial design. For example, you might need to update it because the address in the footer has changed or because you want to add a slogan to the header.

Note

When you change a partial design, the changes are updated on every page that uses this partial design. Therefore, you should make sure that you check where it is used before you change it.

To change a partial design in the Experience Editor:

1. On the ribbon, on the Experience Accelerator tab, click Partial Design.
2. Select the partial design that you want to change.

Note

When you hover over the partial designs, the renderings that belong to this partial design are highlighted in green on your page.

3. Make the changes by for example changing the text or adding renderings

from the Toolbox to the partial design.

4. On the main ribbon, click Save  . Now the partial design has changed for every page that uses it.

Front end

SXA lets back-end and front-end developers work in parallel. Front-end developers can style sites and synchronize changes using Creative Exchange. Learn more about Creative Exchange, The Asset Optimizer and the grid settings:

- [Change a site design using Creative Exchange](#)
- [Import and export a web design with Creative Exchange](#)
- [Modify your site design with Creative Exchange](#)
- [Style SXA sites with Sass](#)
- [Trigger Creative Exchange from your CI server](#)
- [The Asset Optimizer](#)
- [Enable and configure the Asset Optimizer](#)
- [The grid settings](#)
- [The HTML structure of pages and renderings](#)

Working with Creative Exchange

Abstract

Use the Creative Exchange module to style sites with static HTML, CSS and JS files.

The Creative Exchange process is designed to facilitate several different teams working on a website. For example, the team that is working on the theme of the site can work in parallel with other teams.

With Creative Exchange, you can export your site and produce a ZIP file with all the wireframes and all content. The creative agency or the in-house front-end developer can work with the static HTML offline. When the designs are ready, this ZIP file is sent back to be imported again, and the process can continue.



The SXA base theme comes with core CSS and JavaScript. A creative agency or an in-house front-end developer can modify and extend the HTML classes and add CSS/JavaScript assets using their preferred tools. For example, if you added a style to a particular rendering in the HTML, the import process will identify this new class and apply it to the system.

In SXA, any assets, such as images, fonts, and files that are normally referenced within the stylesheet, are considered to be part of the theme and they are imported with your CSS changes.

Note

If you are familiar with JavaScript and Gulp, you can consider using Creative Exchange Live to shorten the time you use changing the design of your site.

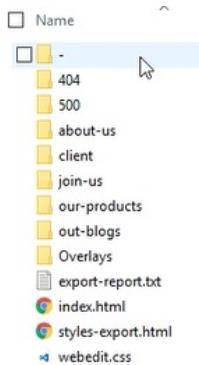
Change a site design using Creative Exchange

Abstract

Site themes consist of the script and styles for all renderings.

When a site is being developed, Creative Exchange enables teams to work together on a site simultaneously. A ZIP file exported using Creative Exchange contains the content, structure, assets, and theme of the site.

You can download this ZIP file and work on the design of the site from your local folder before exporting it back to the team. This lets content authors work on content while the designers make changes to the look and feel of the site.



When you have unzipped the site from Creative Exchange, you can make the following changes:

- Add and modify images and files within the Media Library folder.
- Add classes on nodes that have: `<!-- add your css classes here -->">`
- Change images in renderings.

Note

Changes to the HTML structure, deleting existing classes, changing text content, and changing base theme folders will not be imported back to the system and can damage your site.

This topic describes how to:

- [Change an image](#)
- [Change the layout](#)
- [Change the styling of text renderings](#)

[Change an image](#)

You can change the images used in Image and Image Variant renderings by linking to a new image.

To change the image of an Image rendering:

- In the `index.html` file, change the asset in the `` tag. For example:

```
<div class="component image logo">
    <div class="component-content">
        <a href="index.html">
        <!-- component specific mark-up -->
    </div>
</div>
```

Note

For clean CSS class structure, use lowercase, and a dash to join multiple words. Additional classes for renderings should start with the name of the rendering followed by a name that describes the functionality. For example:

```
<div class="component carousel {guid} {guid} {Styles} carousel-
homepage">
```

or:

```
<div class="component navigation {guid} {guid} {Styles}
navigation-main-horizontal">.
```

SXA pages are divided into rows and columns with splitters. Splitters can have their own list of classes.

To style the layout:

- Find the row splitter and add your class.

```
<div class="row {guid} {guid} {Styles1} <!-- ADD YOUR CSS CLA
    <!-- components mark-up -->
</div>
<div class="row {guid} {guid} {Styles2} <!-- ADD YOUR CSS CLA
    <!-- components mark-up -->
</div>
```

Note

Additional classes for row and column splitters should start with the `column` or `row` prefix, followed by a name that describes the functionality. For example: `<div class="row row-logo">`

- Find the column splitter and add your class.

```
<div class="alpha grid-3 {guid} {guid} {Styles1} <!-- ADD YOU
    <!-- components mark-up -->
</div>
<div class="omega grid-5 {guid} {guid} {Styles2} <!-- ADD YOU
    <!-- components mark-up -->
</div>
```

Note

Classes that you added and imported back into SXA are available for use on other instances.

Change the styling of text renderings

You can change the styling of the text renderings by adding a class.

For example, if you want to change the styling of a Rich Text rendering and the Page List rendering:

1. Open the `index.html` file and navigate to the instance of the rendering.
2. Add the class. For example, add the classes `highlighted` and `hero`:

```
</div>
</div>
<div class="omega grid-8 {DC8D46EB-BCD6-45FF-ADE7-C8541FA2EB5C} {C3AB4DC4-B97C-4B96-8B12-2A0B9E437F9C} {Styles2} add-your-css-classes-here">
    <div class="component rich-text {DC8D46EB-BCD6-45FF-ADE7-C8541FA2EB5C} {8A1DAEFD-5386-45F3-AF8D-3A4ACC1FB26D} {Styles} add-your-css-classes-here highlighted">
        <p>SXA is awesome!</p>
    </div>
</div>
<div id="content" class="main clearfix">
    <div class="row">
        <div class="grid-12">
            <div class="component page-list {6CF97061-4B88-43BB-8EB8-FB6FCB9A5988} {DA9EB08C-26DC-4BE3-A839-EF76C3556E7} {Styles} add-your-css-classes-here hero">
                .. <p>SXA IS awesome!</p>
            </div>
        </div>
    </div>
</div>
```

3. Add the new CSS class and the styling to the main .css file:

```
margin-left: 0; } }
.rich-text.highlighted {color: red}
.page-list.hero [background-color: black;color: white;]
```

Open the `index.html` file to preview your changes on the local instance of the site.

Modify your site design with Creative Exchange Live

Abstract

Make changes to themes and other content and synchronize to the Sitecore environment immediately.

Creative Exchange Live is an SXA feature that enables front-end developers to modify themes and other site content without having to import the files back into the site. Creative Exchange Live works with Gulp tasks that enable you to make changes to themes and other content and synchronize to the Sitecore environment immediately.

If you are familiar with JavaScript and Gulp, Creative Exchange Live shortens the time you use changing the design of your site.

To be able to work with Creative Exchange Live:

- Go to Website\App_Config\Include\z.Feature.Overrides and enable the z.SPE.Sync.Enabler.Gulp.config.disabled file.

Note

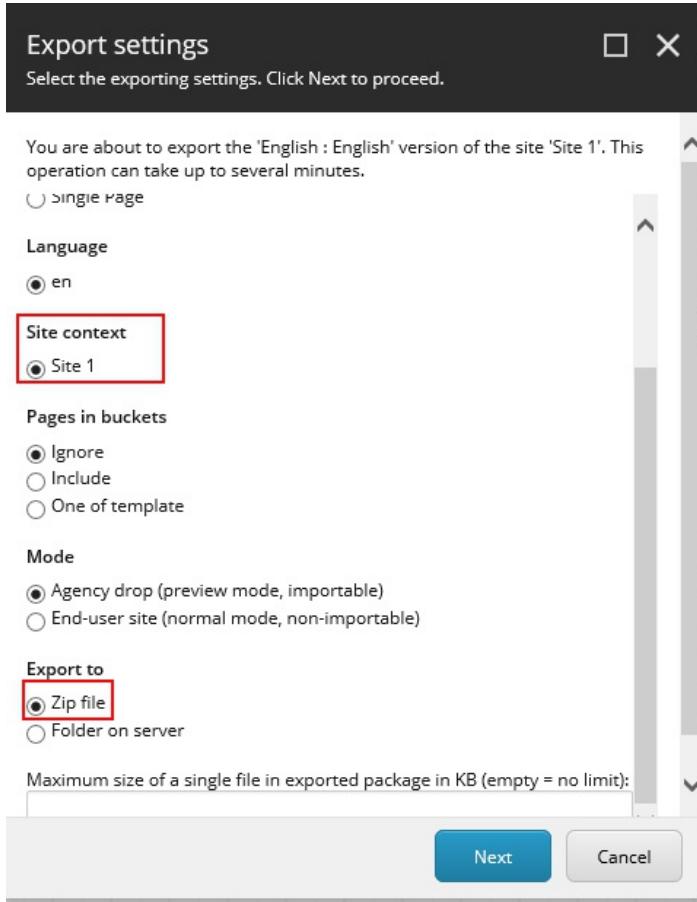
To use Creative Exchange Live, your site must use the Basic 2 theme or a theme based on the Basic 2 theme. This theme contains the Gulp folder.

To use Creative Exchange Live to modify your site design:

1. In the Experience Editor, on the ribbon, on the Experience Accelerator tab, click Export.

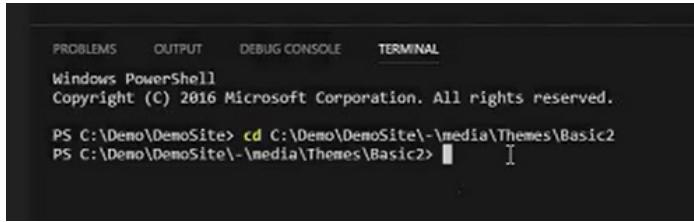
Alternatively, to export a web design in the Content Editor, click a site page and then on the ribbon, on the Home tab, click Export.

2. In the Export settings dialog, edit the settings to download the whole site and in the Export to field, select ZIP file, and then click Download zipped site.



3. Depending on the destination you selected, the ZIP file is available for download or the files are stored in a folder on your server. If you selected the ZIP file option, extract the ZIP file.
4. Navigate to site/Media/Project/Your Tenant/Your Site/Your Theme and open it in Visual Studio Code.
5. In Visual Studio Code, you can open an integrated terminal, initially starting at the root of your workspace. To open the terminal, go to View and click Integrated Terminal.
6. In the explorer pane, right-click Basic2 and click copy path.

7. Change the working folder to the Basic2 folder. In the terminal, enter CD and paste the copied path.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

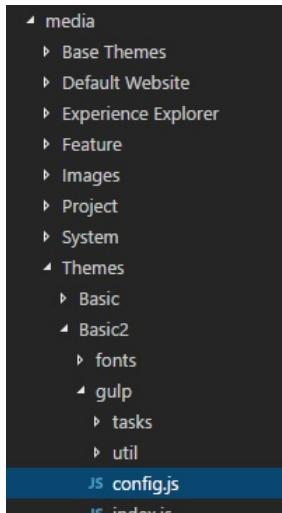
PS C:\Demo\DemoSite> cd C:\Demo\DemoSite\-\media\Themes\Basic2
PS C:\Demo\DemoSite\-\media\Themes\Basic2>
```

8. Use npm and the install command to install your package: npm install

Note

You must have Node.js (Node) installed onto your computer before you can install Gulp.

9. Navigate to Themes\Basic2\gulp\config.js and change the server name to the site that you are working on.



```
media
  Base Themes
  Default Website
  Experience Explorer
  Feature
  Images
  Project
  System
  Themes
    Basic
    Basic2
      fonts
      gulp
        tasks
        util
      config.js
      index.js
```



```
serverOptions: {
  server: 'http://sxa', //need to be changed
  removeScriptPath: '/-/script/v2/master/RemoveMedia',
  uploadScriptPath: '/sitecore modules/PowerShell/Services/RemoteScriptCall.ashx',
  updateTemplatePath: '/-/script/v2/master/ChangeTemplate',
  mediLibraryPath: '/-/script/media/master'
```

Important

Please note that with every Creative Exchange export, your gulp files are overwritten.

To avoid that, you should upload the edited \gulp\config.js file to Sitecore after Creative Exchange Live is configured. You can do this by either uploading a file to Sitecore manually or running a dedicated gulp task that will automatically upload the file once it is modified (watch-gulp). Refer to the `readme.md` file to learn more about other watchers.

10. Run the `gulp watch-html` task. The Gulp watcher takes the Html files of your site.

Note

Go to the `readme` file in the root folder of the theme to see a list of all tasks.

11. Enter your credentials to log in.

Now you can run the Gulp task that will make changes to your site design that are immediately synchronized to your site.

For example, you can add a class to style a rendering and, immediately after refreshing the page, the style is available for selection.

```
<main>
  <div id="content" class="container">
    <div class="row">
      <div class="component promo add-your-css-classes-here new-class col-xs-12" data-component-id="1">
        <div class="component-content">
          <div></div></div>      </div>
        </div>
      </div>
    </div>
```

Style SXA sites with Sass

Abstract

Use Sass to style your SXA site.

Sass (syntactically awesome stylesheets) is an extension of CSS. You can use Sass to style your SXA site. For example, if you want to set global styling for your site, apply specific styling to a rendering variant, or tweak the styling of a breadcrumb separator.

Every SXA theme contains a sass folder that you can use to apply styling. [Creative Exchange Live](#) works with Gulp tasks that enable you to make changes to themes and other content and synchronize to the Sitecore environment immediately. When you use Gulp tasks to modify your site, SXA automatically compiles the Sass files to standard CSS.

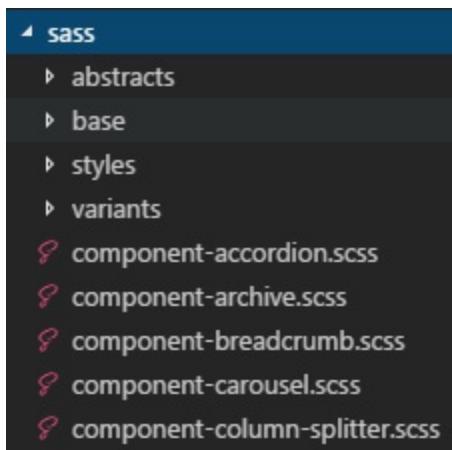
To style sites with Sass:

1. Before you can style sites with Sass, you must ([create](#), [assign](#) and) [export](#) your custom theme.
2. Depending on the destination you selected, the ZIP file is available for download or the files are stored in a folder on your server. If you selected the ZIP file option, extract the ZIP file.
3. Navigate to site/Media/Project/Your Tenant/Your Site/Your Theme and open it in Visual Studio Code.
4. In Visual Studio Code, you can open an integrated terminal, initially starting at the root of your workspace. To open the terminal, go to View and click IntegratedTerminal.
5. Use npm and the install command to install your package: `npm install`

Note

You must have Node.js (Node) installed onto your computer before you can install Gulp.

- Now you can start working with the Sass files. When you save the Sass file, changes are applied to the theme immediately. The sass folder contains the following root files and folders:



Sass file or folder

Description

Contains all Sass tools and helpers that set the global styling for your site. You must put every global variable (for colors, font size, and margins) here. For example, if you want to change the color of the title:

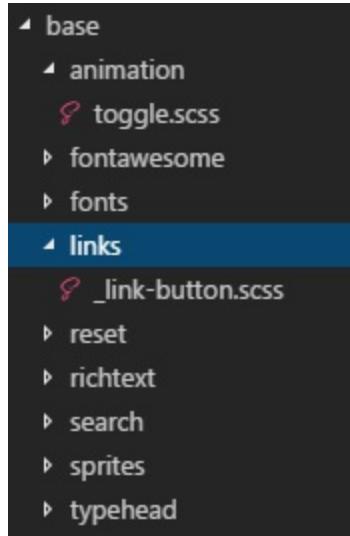
abstracts

```
▲ sass
  ▲ abstracts
    ▲ vars
      ⚡ _colors.scss
      ⚡ _fontSizes.scss
      ⚡ _margins.scss
      ⚡ _functions.scss
      189  $tab-container-bg:$trans
      190  $tab-container-border:$trans
      191  //Title
      192  $title-bg:transparent;
      193  $title-color:#red;
      194  $title-color-active:$text-color
      195  //Toggle
      196  $toggle-header-bg:$bg-black
```

Contains the boilerplate code for the site, typography and a stylesheet that defines the standard styles for commonly used HTML elements. You place Sass fi

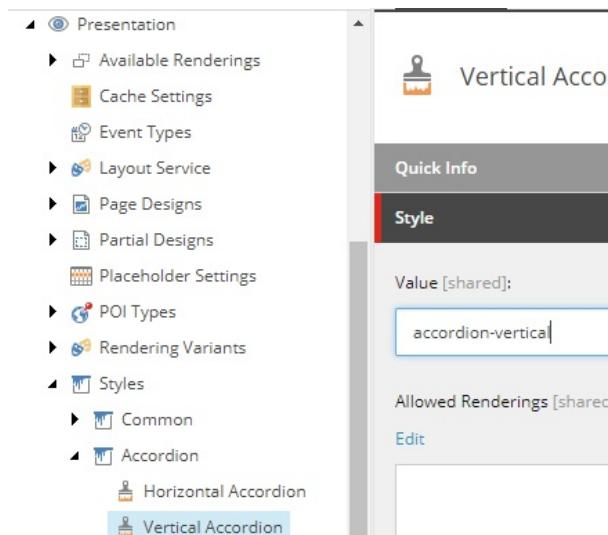
to add styles for more renderings. For example, to set a font style for all renderings or to set the link style.

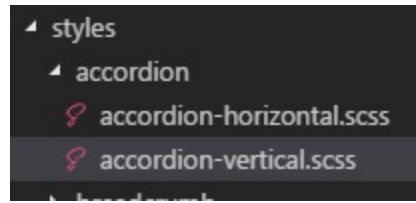
base



Contains the styles for specific renderings, for example to add a class for the accordion rendering. These styles should be the same as the styles used in the Presentation folder of the site
(sitecore/Content/Tenant/Site/Presentation/`)

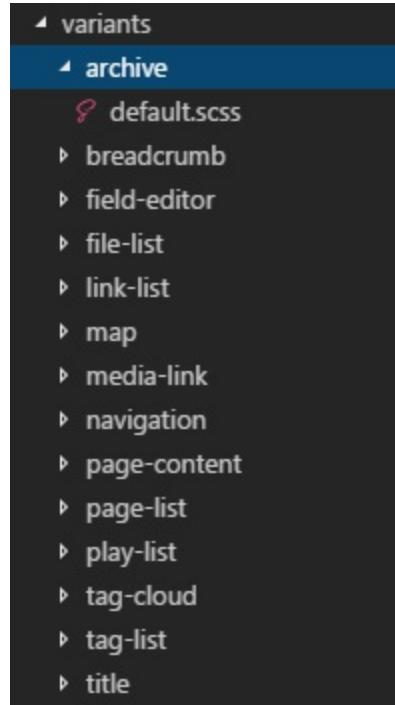
styles





Contains additional styling for rendering variants. You can change the default CSS styles to make styling changes for rendering variants.

variants



All self-contained renderings in their own .scss part. For example, you may need to change the maximum width of the map rendering:

renderings

```
@import "core/colors";
@import "core/mixins";

.mapbox {
    max-width: 100%;
    overflow: hidden;
    position: relative;
    background: $background-color;
    img {
        max-width: none;
    }
    @include respond-to(all-mobile) {
        clear: both;
        margin-top: 30px;
    }
}
```

Output file that brings together all styling files that define global styles for your site.

main.scss

```
1  @import "compass/css3";
2
3  @import "core/colors";
4  @import "core/mixins";
5  @import "core/fonts";
6  @import "core/animations";
7  @import "core/base";
8  @import "core/grid";
9  @import "core/header";
10 @import "core/content";
11 @import "core/footer";
12 @import "core/inputs";
13 @import "core/utils";
```

cookie-notification.scss To determine how cookie notifications display.

overlay.scss

To determine how pop-ups work for your site.

Trigger Creative Exchange from your CI server

Abstract

Trigger Creative Exchange from your CI server

To quickly change the styling of your SXA sites, you can trigger Creative Exchange from your Continuous Integration (CI) server. Continuous integration enables you to integrate code changes routinely into the main branch of a repository, and to test the changes, as early and often as possible.

Important

To trigger Creative Exchange from your CI server, you must have version 1.7.1 of SXA.

This topic describes how to:

- [Trigger export and import](#)
- [Change export options using the Get-CreativeExchangeExportResponse cmdlet](#)
- [Change import options using the Get-CreativeExchangeImportResponse cmdlet](#)

Note

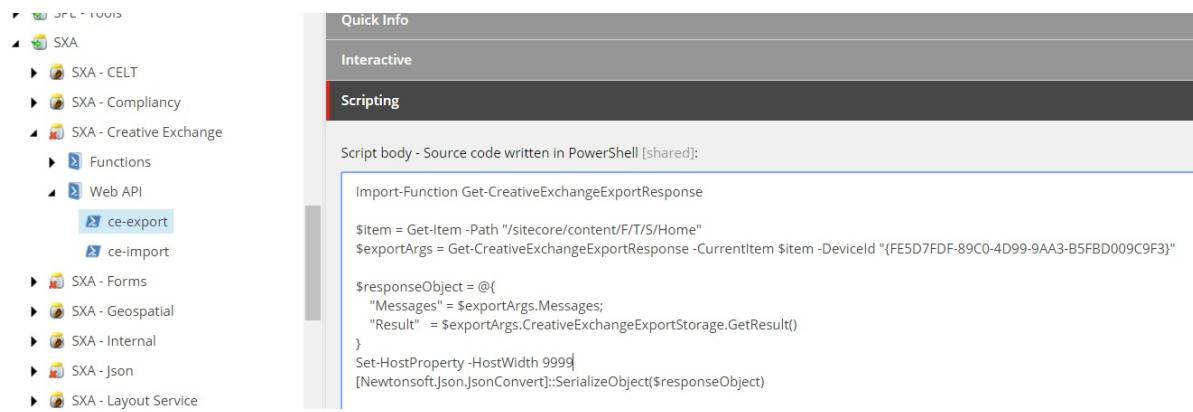
Before you start working on automating your Creative Exchange deployment, you must understand how to use the [Sitecore PowerShell Extensions Web API](#).

[Trigger export and import](#)

To trigger export and import using a simple web request:

- Create endpoints for the Creative Exchange import and export. For examples, go to:

```
/sitecore/system/Modules/PowerShell/Script Library/SXA/SXA -  
/sitecore/system/Modules/PowerShell/Script Library/SXA/SXA -
```



Note

Do not save your web request items under any of the SXA roots as they might be overwritten in the next release.

1. In the import section, import the cmdlets:

```
Get-CreativeExchangeExportResponse
```

```
Get-CreativeExchangeImportResponse
```

2. Specify the path. For example:

```
$item = Get-Item -Path "/sitecore/content/Tenant/Site/Home"
```

3. Specify the cmdlet and its parameters. For example:

```
$exportArgs = Get-CreativeExchangeExportResponse -  
CurrentItem $item -DeviceId "{FE5D7FDF-89C0-4D99-9AA3-  
B5FBD009C9F3}"
```

You can modify the parameters to change the [import/export](#) options.

4. When the export/import is complete, you get the \$exportArgs/\$importArgs object as a result:
 - \$exportArgs
(Sitecore.XA.Feature.CreativeExchange.Pipelines.Export.Ex
 - \$importArgs
 - (Sitecore.XA.Feature.CreativeExchange.Pipelines.Export.Ex
5. Use the \$responseObject object to retrieve the content for the parameters.

You can use it to fetch necessary information and return it as a Web API response. In the following example, we have built a dynamic object and serialized the specified object to JSON format:

```
Import-Function Get-CreativeExchangeExportResponse
$item = Get-Item -Path "/sitecore/content/Tenant/Site/Home"
$exportArgs = Get-CreativeExchangeExportResponse -CurrentItem $item
$responseObject = @{
    "Messages" = $exportArgs.Messages;
    "Result"   = $exportArgs.CreativeExchangeExportStorage.Get
}
Set-HostProperty -HostWidth 9999
[Newtonsoft.Json.JsonConvert]::SerializeObject($responseObject)
```

Change export options using the Get-CreativeExchangeExportResponse cmdlet

The following table describes the available Get-CreativeExchangeExportResponse cmdlet parameters.

To change the export options:

- Modify the parameters of the Get-CreativeExchangeExportResponse cmdlet.

Type

[Item]

[Sitecore.XA.Feature.CreativeExchange.Storage.CreativeExchangeImportResponse]

[Sitecore.XA.Feature.CreativeExchange.Enums.MarkupMode]

[Sitecore.XA.Feature.CreativeExchange.Enums.ExportScope]

[Sitecore.XA.Feature.CreativeExchange.Enums.BucketExportMode]

[string]

[int]

[Sitecore.Data.ID]

[string]

[Change import options using the Get-CreativeExchangeImportResponse cmdlet](#)

The following table describes the available Get-CreativeExchangeImportResponse cmdlet parameters.

To change the import options:

- Modify the parameters of the `Get-CreativeExchangeImportResponse` cmdlet.

Type

[Item]

[Sitecore.XA.Feature.CreativeExchange.Storage.CreativeExchangeImportResponse]

[int]

[Sitecore.Data.ID]

Import and export a web design with Creative Exchange

Abstract

Creative Exchange allows you to import and export site designs.

[Creative Exchange](#) is an SXA feature that enables front-end developers to work on static HTML, CSS, and JS files. You can export an SXA site or even a single page to send to your designers while content editors can continue working on the site. The export process creates a ZIP file of a page or the whole site with its theme and content included. Designers can [modify the files](#) to modify the site's look and feel. Once the design is ready, you can use the tool to import it back into the site.

This topic outlines how to:

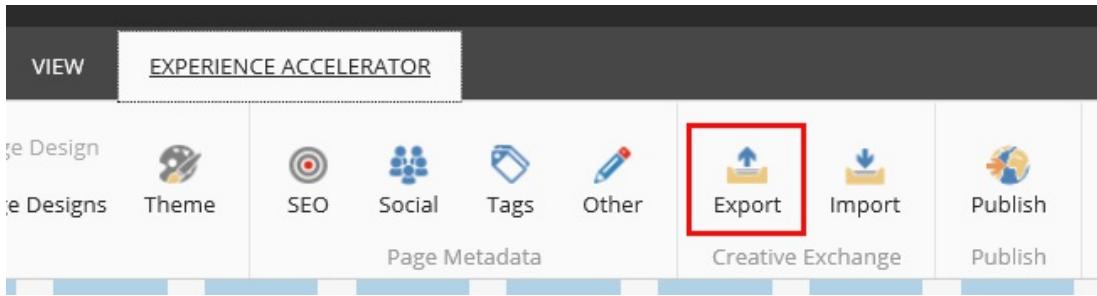
- [Export a web design](#)
- [Import a web design](#)

[Export a web design](#)

To make the site available for designers to work on, you can export the site's wireframes and content using Creative Exchange.

To export a web design using Creative Exchange:

1. In the Experience Editor, on the ribbon, on the Experience Accelerator tab, click Export.



To export a web design in the Content Editor, click a site page and then on the ribbon, on the Home tab, click Export.

2. In the Export settings dialog, edit the settings:

Field	Description
Device	Specify the device you want to export the site design to. If the device is not available, the layout may not be defined in Sitecore. Ask your Administrator for more information.
Exported Content	<p>Specify what you want to export:</p> <p>Single page – exports only the page you are editing.</p> <p>Branch – exports the current page with all its descendants.</p> <p>Site – exports all pages from the current site.</p>
Note	For large sites, you should export the pages of the site separately.
Language	Select the language.

Site context You can use the same page in different sites. Select the site for which you want to export the page.

Specify that you want to export items from item buckets.

Ignore – excludes items from item buckets.

Include – includes items from item buckets.

Pages in buckets One of template – includes one item of each template from item buckets.

Note

Because item buckets can contain large number of pages, do not select this option unless it is explicitly required.

Select how you want to export the content:

Agency drop (preview mode, importable) – exports the pages almost identically to those viewed by visitors, but with some additions to facilitate importing them back into the system.

Mode Author mode (edit mode, importable) – exports the pages from the Experience Editor containing additional markup that enables on-page editing.

End-user site (normal mode, non-importable) – exports the pages identically to those viewed by visitors but cannot be imported back into your site.

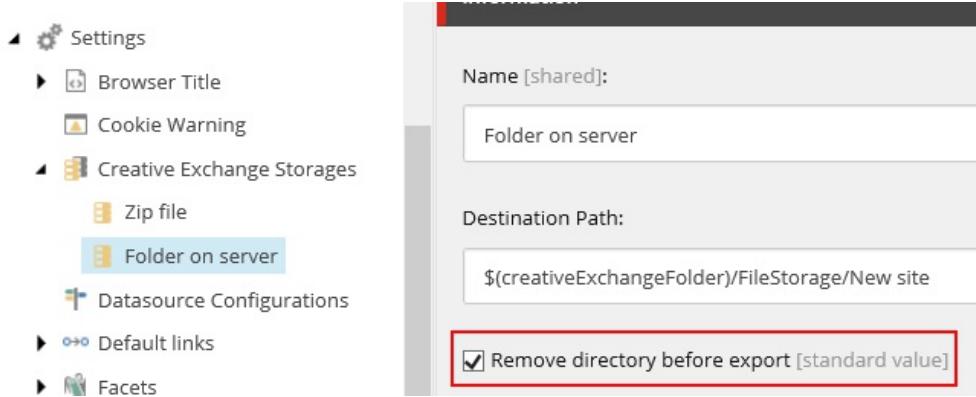
Specify the destination for your exported content:

Zip file – creates a package of compressed files.

Export to Folder on server – saves all the exported files directly to the server file system, where they can be managed by your revision control system. This can be convenient if you are working with external agencies and need to access the files through SVN or Git.

Maximum size of a single file in exported package Enter a number to limit the size of zipped export files. This can be convenient when your site contains large assets, such as videos.

If you have many zip files from previous exports, you can remove these by selecting Remove directory before export in SITENAME/Settings/Creative Exchange Storages/Folder on server .



3. Click Next to start the export.

Depending on the destination you selected, the ZIP file is available for download or the files are stored in a folder on your server. You can configure the path to store exported sites in the Destination Path field. Go to: [Sitename]/Settings/Creative Exchange Storages/Folder on server

Note

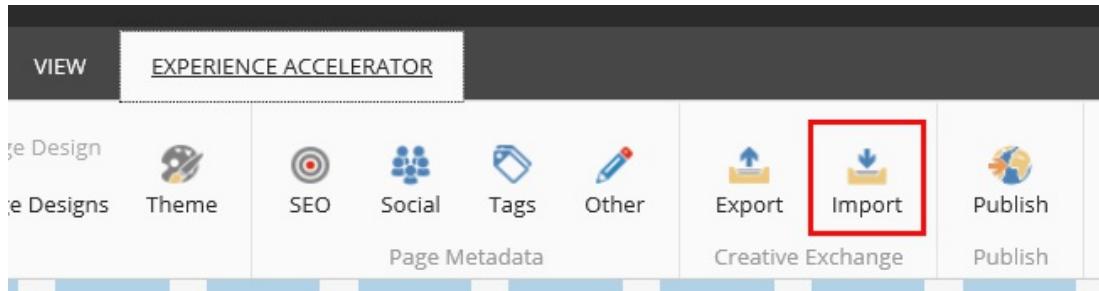
Possible reasons for your export to fail are: pages could not be rendered due to errors, linked assets are missing, or a Creative Exchange timeout.

[Import a web design](#)

Once the web design work is done, the folder can be zipped and imported back into the site using Creative Exchange.

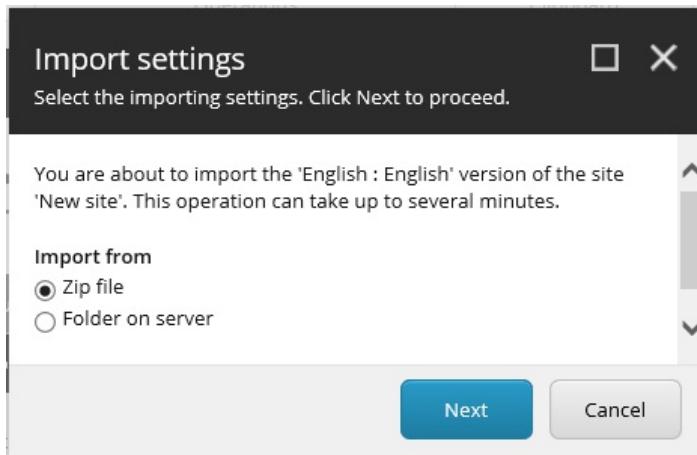
To import a web design into the site:

1. In the Experience Editor, on the Experience Accelerator tab, click Import.



To import a web design in the Content Editor, click a site page and then on the ribbon, on the Home tab, click Import.

2. In the Import settings dialog box, select the source of the designs and click Next.
3. Depending on the source that you selected, follow the steps in the wizard, and then click Next.



4. You receive a confirmation message when the import is complete.

Note

Possible reason for your changes not to be reflected after import: the theme you changed is part of a restricted location.

Styling

Optimize CSS styles and JS scripts by using the Asset Optimizer. Learn more about the grid settings and the HTML structure of pages and renderings:

- [The Asset Optimizer](#)
- [Enable and configure the Asset Optimizer](#)
- [The grid settings](#)
- [The HTML structure of pages and renderings](#)

Enable and configure the Asset Optimizer

Abstract

Enable the Optimizer globally for entire Sitecore instance or locally for selected tenants.

The Asset Optimizer is a module that optimizes CSS styles and JS scripts. When it is enabled in a production environment, the Asset Optimizer improves overall site performance by reducing the amount of data that needs to be transferred. Sitecore administrators can enable this module either globally for the entire Sitecore instance or locally for selected tenants.

Note

It is best practice to disable the Asset Optimizer in development environments and to enable it in production environments.

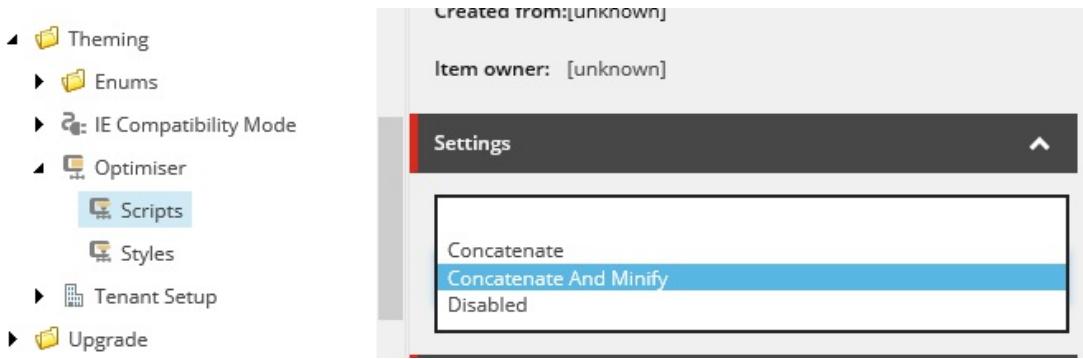
This topic describes how to:

- [Enable the Asset Optimizer globally](#)
- [Change the optimization settings for a specific site](#)

[Enable the Asset Optimizer globally](#)

To enable the optimizer globally:

- In the Content Editor, navigate to /sitecore/system/Settings/Foundation/Experience Accelerator/Theming/Optimiser and for both Scripts and Styles select Concatenate and Minify to minify all files and concatenate them into one file.



Note

Changes take effect immediately after the item is saved. Anonymous users might not see the changes until you publish.

Change the optimization settings for a specific site

To change the optimization settings for a specific site:

1. In the Content Editor, navigate to `sitecore/content/.../SITE_NAME/Presentation/Page Designs`.
2. In the Asset Optimization section, in the Styles Optimizing Enabled and Scripts Optimizing Enabled fields, to override styles and scripts optimization settings, select:
 - Default – to inherit global settings
 - Yes – to always enable optimization for this site.
 - No – to always disable optimization for this site.

- ▶ Home
- ▶ Media
- ▶ Data
- ▶ Presentation
 - ▶ Available Renderings
 - ▶ Event Types
 - ▶ Page Designs
 - ▶ Default
 - ▶ Page Designs
- ▶ Partial Designs
- ▶ POI Types
- ▶ Rendering Variants
- ▶ Styles
- ▶ Settings

Default

Assets Optimization

Styles Optimising Enabled [shared]:

- Default

Scripts Optimising Enabled [shared]:

- Default**
- Yes
- No

The Asset Optimizer

Abstract

Improve end user experience by optimizing CSS styles and JS scripts.

In a production environment, the SXA Asset Optimizer improves the end user experience by optimizing CSS styles and JavaScript, and reducing the amount of data that needs to be transferred.

In SXA, renderings and other front-end functionality are styled or scripted in files stored in themes that tend to have the front-end capabilities broken down on a component level. This makes front-end development easier, but the number of files that need to be served may not be acceptable in a production environment. Having a large number of small files causes performance to deteriorate and can also cause issues in older browsers. Therefore, we recommend that you [enable the Asset Optimizer](#) in production environments. Sitecore administrators can enable this module either globally for an entire Sitecore instance or locally for selected sites.

The Asset Optimizer groups assets together according to the following rules:

- Assets that come from the same folder.
- Assets that have the same extension.

Assets from each group are concatenated into one asset and cached on the server. References to individual assets in the markup are replaced with one reference for each group, so that:

- The reference path starts with the same folder path as the initial assets.
- The reference path ends with the same extension as the initial assets.
- The link tag has the same value of media attribute as the initial assets.

The following code sample is a piece of sample markup for just one theme used on the page where each asset is referenced separately:

If the Asset Optimizer is enabled, all the links in the previous HTML code sample are gathered in a separate theme file:

```
<script src="/sitecore/shell/-/media/Base-Themes/SearchTheme/Scri
```

Note

Even with the Asset Optimizer enabled, you can always request a page with the assets broken down into the original files by appending the following parameter to your request URL:

aodisabled=1

The grid settings

Abstract

The SXA grid definition, its settings, and classes.

A grid system is a library of HTML/CSS components that let you structure a website easily. Grid systems help you create responsive websites that have consistent designs and ensure cross-browser support. Understanding how the SXA grid layout works provides you with an insight into how to use and change grid settings, and how to add classes.

This topic describes:

- [The grid systems](#)
- [The grid definition](#)
- [The grid classes](#)

[The grid systems](#)

By default, SXA comes with the following grid systems:

- Bootstrap 3 – responsive, mobile-first grid system with a 12-column layout that includes predefined classes. Default devices are:
 - Phones
 - Tablets
 - Desktops
 - Large desktops
- Bootstrap 4 - responsive, mobile-first grid system with a 12-column

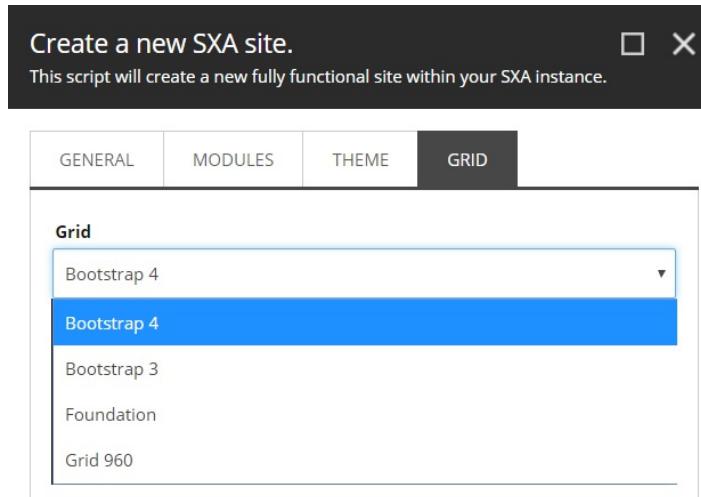
layout that includes predefined classes. Default devices are:

- Compact phones
 - Phones
 - Tablets
 - Laptops
 - Desktops
- Foundation – responsive, mobile-first grid system with 12-column flexible grid that can scale out to an arbitrary size. Default sizes are:
 - Small
 - Medium
 - Large
 - Grid 960 – 12-column grid, based on a width of 960 pixels. Default settings are:
 - Size
 - Prefix
 - Suffix

Note

All grid systems have different system rules, classes, and options. For more detailed information, refer to the different grid systems own documentation.

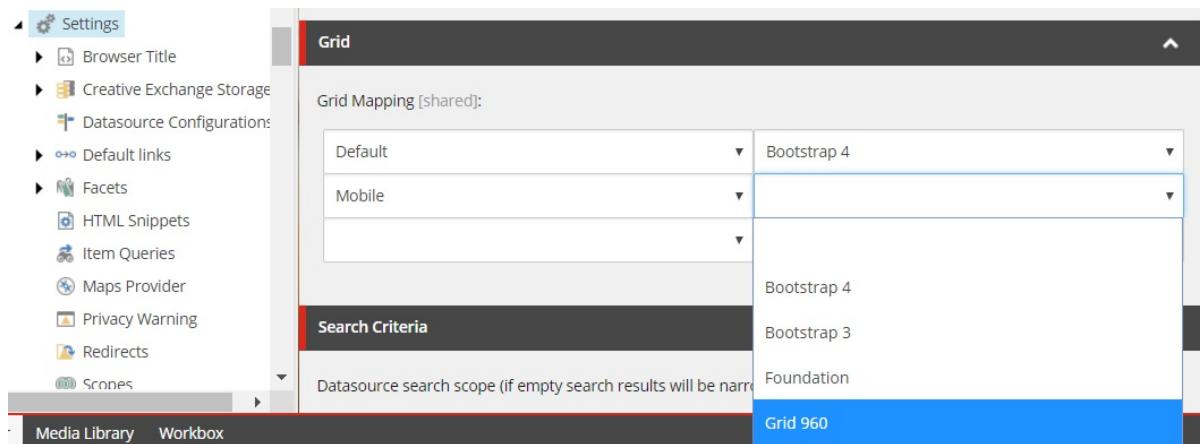
You can select a grid system when you create a site.



Note

If you do not select a grid during the site creation process, by default the Bootstrap grid is applied.

In the site Settings folder (Sitecore/Content/Tenant/Site/Settings), in the Grid section, in the Grid Mapping field, you can map devices to a specific grid system. For example, you can use the Grid 960 system for mobile devices:



Note

It is important to be aware that changing the grid system after you create your site requires many manual changes. Because of the references on your pages to the previous grid system, making a change to the grid system will break

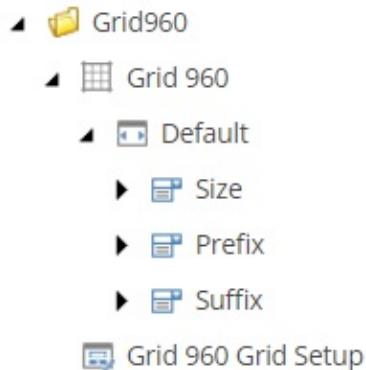
your layout.

The grid definition

Grid systems are stored as a feature in Sitecore (sitecore/System/Settings/Feature/) and include:

- The Grid Definition item – includes all grid system specific items (the devices and their classes).
- The Grid Setup item – the scaffolding item to include the grid in the drop-down list of the Create a new Experience Accelerator site wizard.

The Grid Definition item includes all grid system specific items that a content editor can use in the Experience Editor.



The following fields are included in the Grid Definition item:

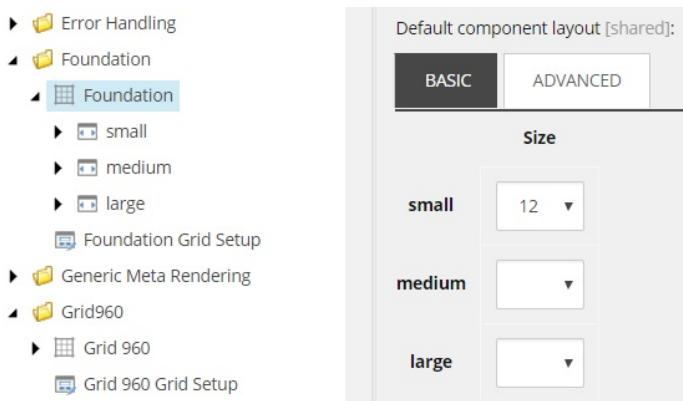
Field	Description
Name	The name of the grid system as you want it to display in the site creation wizard.
Rendering	

parameters	Determines the way the grid parameters are rendered.
field type	
Default component layout	Sets the default column size for renderings that you drag on the page.
Grid theme	Links to the theme used for the grid.
Grid field parser type	Determines the parser type to parse the grid fields.
Grid body view path	Defines the body layout of a page.
Flex	List of CSS classes that determines the way placeholders behave when they are not set to a fixed size. For example, <code>row expanded</code> .
Fixed	Set of CSS classes that determines the way placeholders behave when they are fixed. For example, <code>row</code> .
Row class	CSS class added to the DIVs generated for the Row Splitter rendering.
Row Container Class	CSS class added to the parent DIV of all rows.

The Grid Setup item includes the following fields:

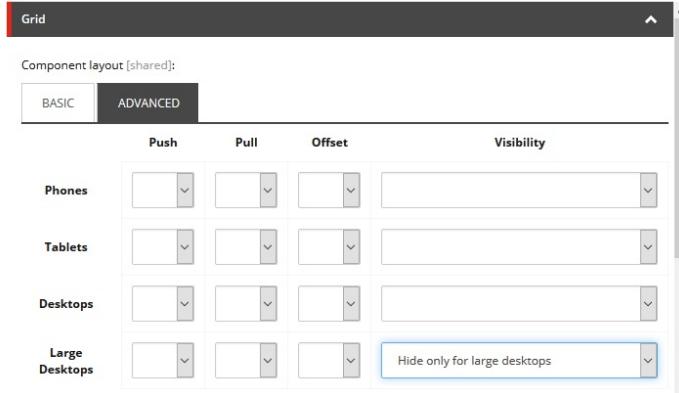
Field	Description
Name	The name of the grid system as you want it to display in the site creation wizard.
Dependencies	Lets you determine the order in which the features are installed.
Grid Definition	Refers to the Grid Definition item. For example, for the Foundation grid system: Settings/Feature/Experience Accelerator/Foundation/Foundation

You can change the default layout settings of the renderings in the grid definition item of the grid system. For example, for the Foundation grid (sitecore/System/Settings/Feature/Experience Accelerator/Foundation):

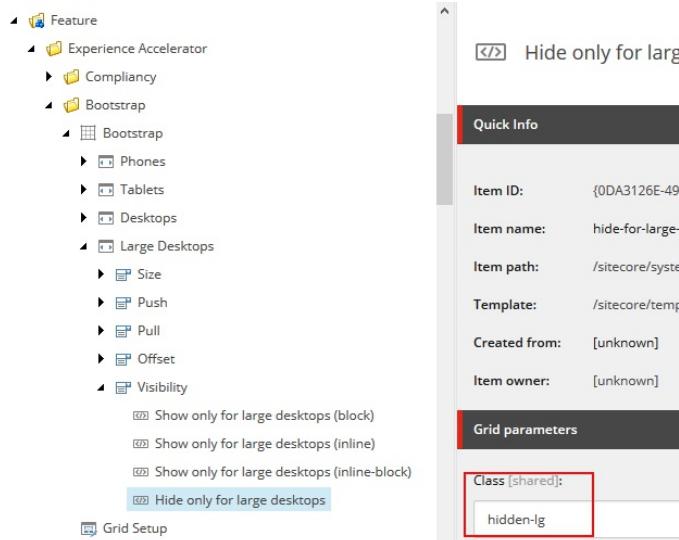


The grid classes

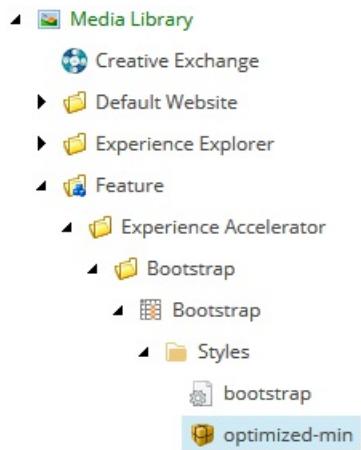
The classes defined in the Grid Definition item correspond to the grid system details in the Media Library. For example, in the Experience Editor, you have the option to hide a rendering for visitors using a large desktop.



This option is set in the Visibility settings of the Bootstrap grid system (sitecore/System/Settings/Feature/Experience Accelerator/Bootstrap/Bootstrap/Large Desktops/Visibility/Hide only for large desktops):



This class is taken from the bootstrap.css file loaded in the Media Library (sitecore/Media Library/Feature/Experience Accelerator):



The HTML structure of pages and renderings

Abstract

SXA uses a standardized HTML structure to rely on a consistent HTML rendering.

SXA separates structure (HTML) from design (CSS) to make it easier to change the design of websites. To make this possible, SXA provides a stable well-structured HTML code that is the same for every page. Users can apply different styling without changing the underlying code.

This topic describes the basic structure for:

- [Page HTML](#)
- [Rendering HTML](#)
- [JavaScript](#)
- [CSS](#)

[Page HTML](#)

All SXA pages use the following layout structure:

```
<html>
<head>
    <!-- meta renderings placeholder "head" -->
</head>
<body>
    <!-- meta renderings placeholder "body-top" -->
    <div id="wrapper">
        <div id="header" class="main clearfix">
            <!-- header components -->
```

```

</div>
<div id="content" class="main clearfix">
    <!-- content components -->
</div>
<div id="footer" class="main clearfix">
    <!-- footer components -->
</div>
</div>
<!-- meta renderings placeholder "body-bottom" -->
</body>

```

All regular renderings can be placed in the following containers:

- Header
- Content
- Footer

Meta renderings can be placed on Meta Partial Designs in the following containers:

- Head
- Body-top
- Body-bottom

Designers can use page splitters to generate additional columns or rows inside the header, content, or footer containers.

There are two types of splitters:

- Column splitters – generates `div`s with proper grid classes wrapped by the row container. Grid values specify the column widths.

```

<div class='row'>
    <div class='grid-6'></div>
    <div class='grid-6'></div>
</div>

```

- Row splitters – generates empty `row` `div`s that fill the full width of the

available parent container.

```
<div class='row'></div>
<div class='row'></div>
<div class='row'></div>
```

You can add classes for both columns and rows. You can also mark specific splitter sections and style them differently than the other sections. This can be useful for styling a page part that breaks the grid system. However, you can only override grid behavior for specific rows.

Rendering HTML

All SXA renderings are designed to be easily styled. Because the HTML is very standard, it is easy for theme developers to apply CSS and JavaScript.

The following example uses the Accordion rendering. The HTML structure for renderings is wrapped by `div` with the component class and the component name `accordion`. You can add CSS class variants for styling purposes.

```
<div class="component accordion {custom classes}" data-properties
  "easing":"easeInOutBounce", "canOpenMultiple":false, "canToggle":fa
>
  <div class="component-content">
    <div>
      <ul class="items">
        <li class="item">
          <div class="toggle-header">
            <div class="label">
              Header content
            </div>
          </div>
          <div class="toggle-content">
            Section content
          </div>
        </li>      </ul>
      </div>
    </div>
  </div>
</div>
```

Note

Use classes for styling. Do not use IDs. Div IDs are used by Sitecore and cannot be changed.

The following properties are used by JavaScript to control rendering behavior:

- expandedByDefault – first accordion tab is visible.
- expandOnHover – expand tab on mouse enter and close on mouse leave events (the canOpenMultiple property is not used in this case).
- canOpenMultiple – open multiple tabs at the same time (the canToggle property is always active and cannot be disabled).
- speed and easing – can be used everywhere to define transition method and time.

Every SXA rendering contains the same wrapping structure:

```
<div class='component <component-name>'>
    <div class='component-content'>
        <!-- component html -->
    </div>
</div>
```

The code inside will be different for each rendering. Often, there will be an additional wrapping div with a unique ID that describes the specific rendering used by Sitecore. CSS/JS scripts should not use these IDs.

Note

The inner rendering structure uses clean markup with dash-separated class convention. There are a few exceptions: some elements, such as forms, use additional Sitecore modules (for example, Web Forms for Marketers) and generate HTML that looks different (camelCase class names, and tables in some cases). You cannot modify this HTML.

Every HTML rendering is part of a platform and you cannot change it for a single project or site, except where the HTML is shaped by [rendering](#)

[variants.](#)

JavaScript

Complex renderings have their own JavaScript. These scripts are located in the JavaScript framework in the main themes folder. The framework provides public methods, such as register and init, to register renderings and additional helpers, such as cookies.

All back-end properties used by JavaScript are placed in the data-properties attribute inside the rendering wrapping div. They are encoded in JSON format.

```
<div class='component accordion' data-properties='{"expandOnHover": "easeInOutBounce", "canOpenMultiple": false, "canToggle": false}'>
```

The following is a clipped version of the JavaScript framework:

```
var XA = XA || (function ($, document) {
    var api = {}, onPreInitHandlers, onPostInitHandlers, modules
    onPreInitHandlers = new Array();
    onPostInitHandlers = new Array();
    /*
     * Register new module
     * @params name - name of the module
     * @params api - API object of the module
     * @params init - init function for module, if not defined ap
     */
    api.register = function (name, api, init) {
        modules[name] = {
            name: name,
            api: api,
            init: init || api.init || (function () { })
        };
    };
    api.registerOnPreInitHandler = function (handler) { onPreInitHandlers.push(handler); }
    api.registerOnPostInitHandler = function (handler) { onPostInitHandlers.push(handler); }
    var initScheduled = false;
    /*
     * Initializes all registered modules
     */
    api.init = function () {
```

```

        if (!initScheduled) {
            initScheduled = true;
            XA.ready(function () {
                try {
                    for (var name in modules)
                        if (modules.hasOwnProperty(name)) {
                            $xa.each(onPreInitHandlers, function
                                modules[name].init();
                            $xa.each(onPostInitHandlers, function
                            }
                }
                finally {
                    initScheduled = false;
                }
            });
        };
    /*
     * Wrapper around $(document).ready - fires given function wh
     */
    api.ready = function (fn) {
        $(document).ready(fn);
    };
    return api;
})($, document);
XA.init();

```

CSS

You can replace a single class to change rendering behavior, for example, to change from standard navigation to mobile navigation. You can also create your own rendering variant by adding CSS classes and add styling and JavaScript functionality. You must adhere to naming conventions. Classes for specific renderings always start with the name of rendering. Add words that explain the functionality of the new class using the dash name convention. For example: .navigation-mobile, .navigation-main-horizontal, .navigation-sidebar.

Headless

SXA enables you to model your data in JSON (JavaScript Object Notation). Learn more about modeling your SXA data in JSON format:

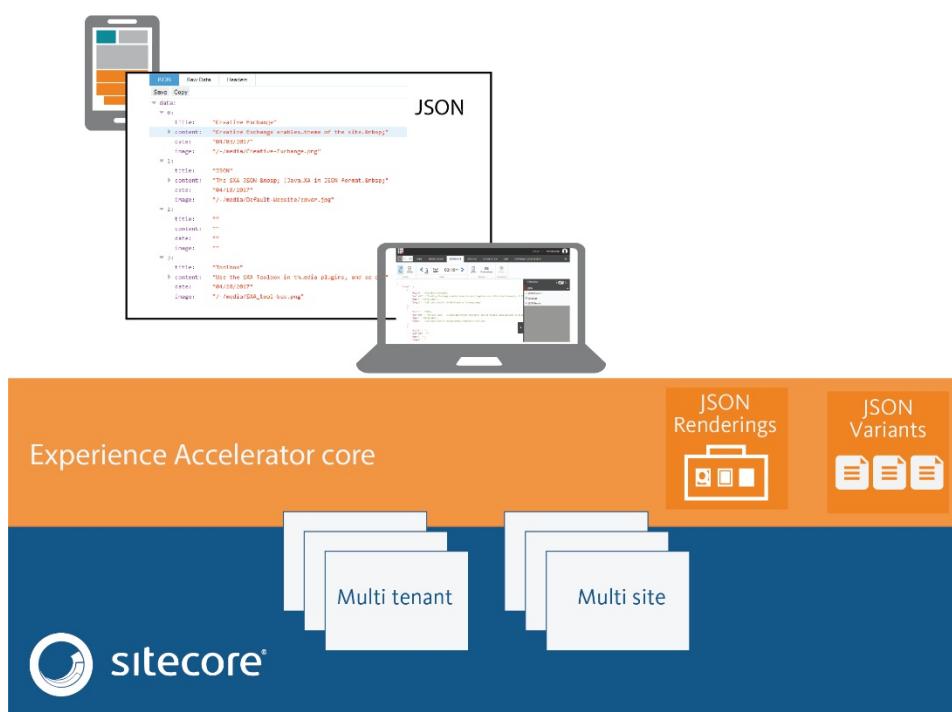
- [Introducing SXA data modeling](#)
- [Add a JSON rendering and variant to your page](#)
- [The JSON search parameters](#)
- [Use the SXA Layout Service to model your pages in JSON](#)
- [Trigger a JSON layout rendering](#)

Introducing SXA data modeling

Abstract

JSON layout, JSON renderings, and JSON variants explained.

SXA enables you to model your data in JSON (JavaScript Object Notation). For example, if you want to build a mobile app and feed it with SXA content, you can edit the JSON content on the page and have output content in JSON instead of HTML.



The JSON feature provides a JSON API to access the SXA content. The content for the site is accessible via a web-service API in JSON format.

This topic describes:

- [Enable SXA data modeling](#)
- [JSON layout](#)

- [JSON renderings](#)
- [JSON variants](#)

[Enable SXA data modeling](#)

By default, JSON functionality is added to your tenants and sites. Both the Create a tenant and Create a new Experience Accelerator site wizards have the JSON feature selected.

The screenshot displays two overlapping dialog boxes from Sitecore's Experience Accelerator (XAP) interface.

Create a tenant (Top Dialog):

- Tenant name:** New tenant
- Features:** A list of checkboxes including Composites, Content Validation, Error Handling, and **JSON** (which is highlighted with a red box).
- Select all | Unselect all | Invert selection**

Create a new Experience Accelerator site (Bottom Dialog):

- GENERAL FEATURES THEME GRID** (Tabs)
- Features:** A list of checkboxes including Compliancy, Composites, Content Tokens, Context, Creative Exchange, Engagement, Events, Forms, Geospatial, and **JSON** (which is highlighted with a red box).
- Select all | Unselect all | Invert selection**
- Compliancy**
- Composites**
- Content Tokens**
- Context**
- Creative Exchange**
- Engagement**
- Events**
- Forms**
- Geospatial**
- JSON**
- Local Datasources**
- Maps**
- Media**
- Navigation**
- Overlays**
- Page Content**

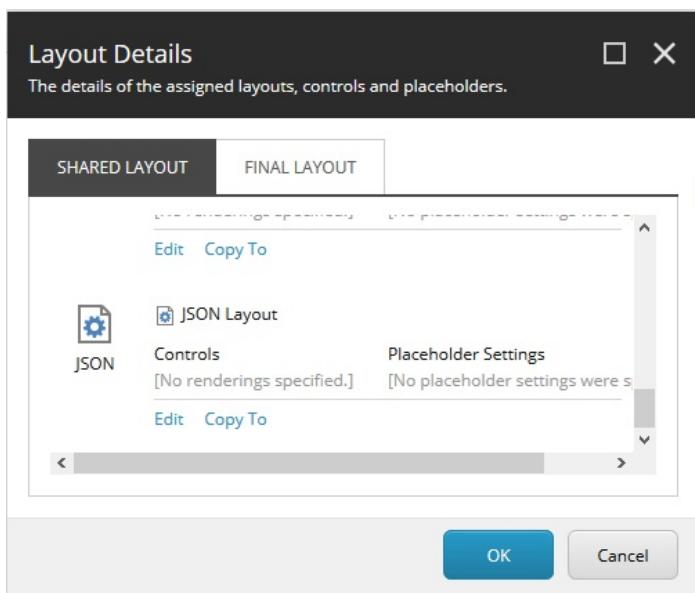
Buttons at the bottom:

- Ok
- Cancel

Before you can start working with the JSON renderings, you must switch to the JSON device.

JSON layout

The SXA JSON device uses a separate layout that is stored in the JSON Layout folder (sitecore/Layout/Feature/Experience Accelerator/JSON Layout). When you switch to JSON device in the Experience Editor, you can see that it is using the JSON layout.



JSON renderings

There are special [renderings that you can use to build JSON layout](#). These renderings are stored in sitecore/Tenant/Site/Presentation/Available Renderings/JSON. Out of the box, there are three JSON renderings available:

- JSON Content – displays fields of a page using JSON renderings and JSON variants.
- JSON List – displays lists of pages by predefined or composed queries.
- JSON Results – displays the results of a search query.

JSON variants

To support additional fields for JSON renderings, you can create JSON variants. JSON variants are stored in the grouping items of the JSON renderings (for example, sitecore/Tenant/Site/Presentation/ Rendering Variants/JSON content).

To define the fields to display, you can add child items to the rendering variant. To do this, you right-click the variant, click Insert, and then click the relevant item.

You can add the following child items to a JSON variant:

- JSON Checkbox – special JSON variant field that renders the values of the fields of type Checkbox in different modes.
- JSON Date – renders date and time in a custom format.
- JSON Field – renders field in HTML.
- JSON File – JSON rendition of a File field that outputs link to a file.
- JSON Image – specialized JSON variant field that renders the value of fields of type Image in different modes.
- JSON Multilist Reference – special JSON field value that provides an enumeration over references defined in a Multilist derived fields. Uses child items to determine what to render. For example, lists all pages with their titles.
- JSON Placeholder – provides a placeholder, for example, to load a JSON list within a JSON content rendering.
- JSON Reference – allows to render content from a linked item. Define the field you want to use as the reference in the PassThroughField field. You can use this variant field for the following fields: LinkField (GeneralLink, DropLink), FileField (File), ImageField (Image), and ReferenceField (Droptree). The reference item needs to contain other

JSON rendering items as its child items. This can be convenient when you have item linked from the rendered item and want to display fields from that item. By adding a name in the Variant details section, you can create an additional JSON object.

- JSON Section – used for grouping of fields. This renderer contains fields such as Name (to create a wrapping element for the children renderers underneath). You can nest Section items to create a more complex variant structure.
- JSON Template – lets you define the NVelocity template used to render part of the JSON. The `getVelocityTemplateRenderers` pipeline is used to add a custom renderer that later on can be used in the NVelocity template.

Note

NVelocity is a .Net-based template engine. It permits developers to use the simple yet powerful template language to reference objects defined in .Net code.

- JSON Text – displays text and is used to render a custom string, of for example, a description. You can use the following fields: Name (define property name for the text), Text (containing the text to be rendered).
- JSON Token – SXA supports the following tokens: \$id (renders the id of the item); \$size (formats the size of the attached asset); \$name (renders the name of the item); and \$FileTypeIcon (renders a span with the CSS class equal to the file extension). The custom `resolveVariantTokens` pipeline is used to extend the set of variant tokens.
- JSON Rich Text – displays Rich Text field in your JSON component. Field can be rendered using one of the provided modes (for example, can be cleaned up before output to strip HTML tags from it).

Depending on the item you add, you can set different fields:

Field	Description
	Depending on the rendering variant child item, the mode field can be set to different views:
Mode	<ul style="list-style-type: none"> • Raw – renders a raw Sitecore value, unformatted data. • Text – renders text. • Rendered – renders the value using the field renderer. • Plain – renders the item as plain text. • Path – renders the path of the item. • Guid – renders the globally unique identifiers of the item in Sitecore.
Value for True	Determines the value that is rendered for true.
Value for False	Determines the value that is rendered for false.
Name	Overrides the JSON property name. By adding a name for the JSON Reference field, you can create an additional object.
Field name	Name of the field current item.
PassThroughField	Defines the name of the field that stores the link to a different item. For example, a simple reference, such as a

	link, or a multi-reference field, such as a Tree list.
Date format	Select the date format from the drop-down list.
	Define the NVelocity template that renders part of the component variant HTML. You can use the following objects:
Template	<ul style="list-style-type: none">• <code>\$item</code> – access to the current item (the <code>\$item.Name</code> object displays the current item name).• <code>\$dateTool.Format(date, format)</code> – formats date and time values.• <code>\$numberTool.Format(number, format)</code> – formats numbers.• <code>\$geospatial.Distance</code> – displays distance to points of interest in geospatial searches.
Text	The text to render.
	Use special tokens to format certain field values. The following tokens are supported:
Token	<ul style="list-style-type: none">• <code>\$Size</code> – displays the size of a file depending on the size unit.• <code>\$FileTypeIcon</code> – displays an icon depending on the file extension.

Add a JSON rendering and variant to your page

Abstract

Construct your pages by dragging JSON renderings from the Toolbox to the page.

The JSON data model comes with three predefined renderings to make page design easy. You can construct your pages by dragging renderings from the Toolbox to the page. All JSON renderings are JSON variants enabled.

By default, there are three JSON renderings available:

- JSON Content – used to expose fields of a page using JSON renderings and JSON variants. By default, this rendering loads two fields on the page. You can edit these fields.
- JSON List – displays lists of pages by predefined or composed queries.
- JSON Results – displays the results of a search query.

This topic outlines how to:

- [Switch to JSON](#)
- [Add a JSON rendering](#)
- [Add a JSON variant](#)
- [Select a JSON variant](#)

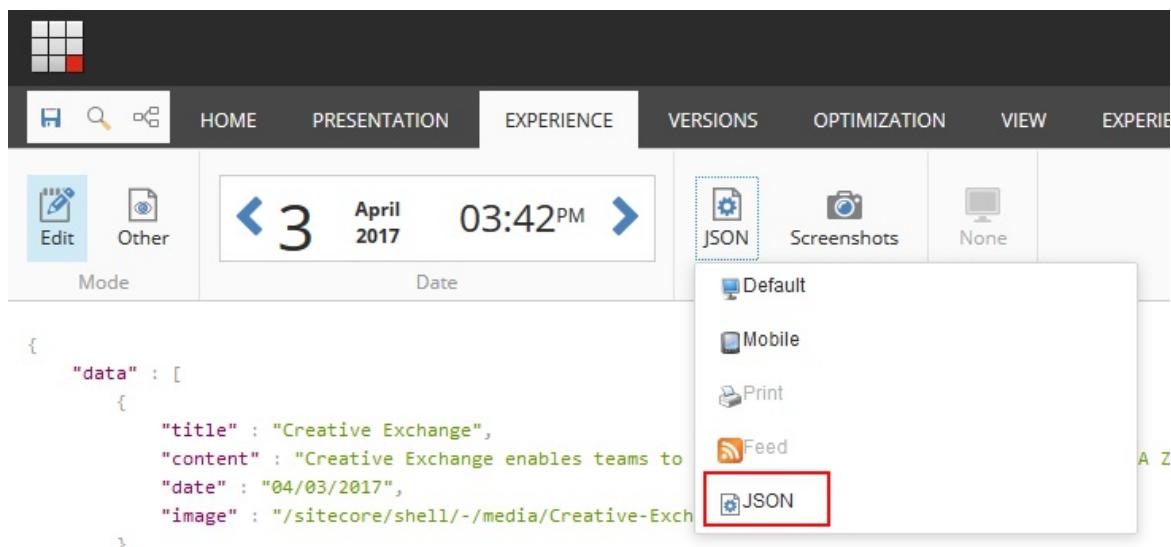
[Switch to JSON](#)

Before you can start working with the JSON renderings, you must switch to

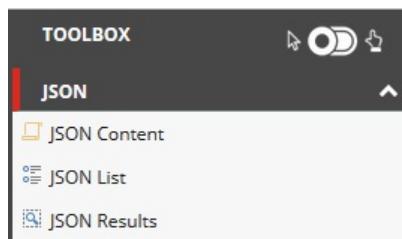
the JSON device.

To switch to the JSON device:

- In the Experience Editor, on the ribbon, on the Experience tab, click Default, and then click JSON.



The JSON renderings are now available for you to use in the Toolbox in the right panel.



Note

If you work on a touch device, such as a tablet or a touch-enabled laptop, by default the Toolbox opens in touch mode. If you work on a touch-enabled laptop, you can switch to desktop mode by clicking the arrow in the upper right of the toolbar .

Add a JSON rendering

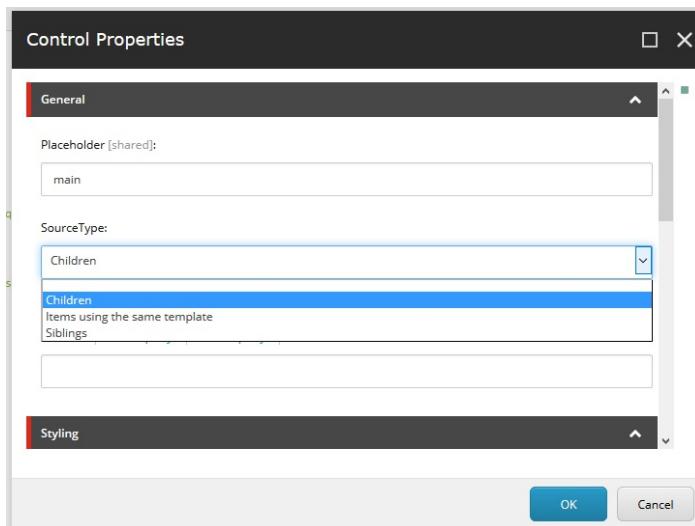
In the Experience Editor, you can add a rendering to a page by dragging it from the Toolbox. For example, if you want to list the pages of your site use the JSON List rendering.

To add a JSON rendering to a page:

1. In the Toolbox in the right panel, find the relevant rendering. When you click and start dragging a rendering, the placeholders where you can drop the rendering light up in blue. Click to drop the rendering on the page. For example, drag the JSON List rendering to the page.

Note

By default, the JSON List rendering displays the children of the current item. For example, when you start on your home page and drag the JSON List rendering to the page, it lists all the pages of your site. This is because the SourceType field by default is set to Children.



2. To see the list, save the page:

```
{
  "data" : [
    {
      "title" : "Creative Exchange",
      "content" : "Creative Exchange enables teams to work tog
    },
    {
      "title" : "JSON",
      "content" : "The SXA JSON (JavaScript Object Notation)
    },
    {

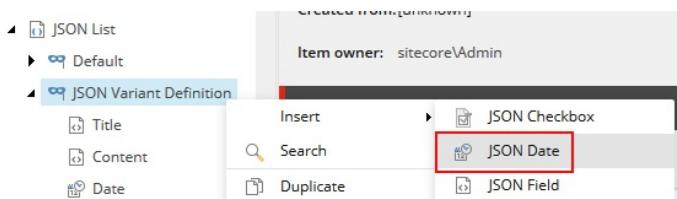
```

Add a JSON variant

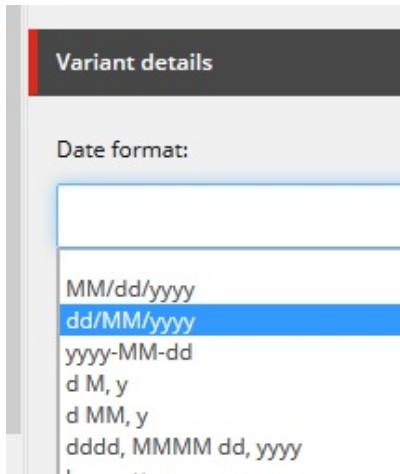
To build a more complex list, you can add a JSON variant. For example, if besides the title and content, you want to add a field for the image and a date.

To add a JSON variant:

1. Navigate to sitecore/Content/Tenant/Site/Presentation/Rendering Variants/JSON List, right-click JSON List, click Insert, and click JSON Variant definition.
2. Enter a name and click OK. For example, to add JSON fields for the title and the content, add a JSON Image, and a JSON Date field.



3. In the Variant details section, select the date format and save the item.



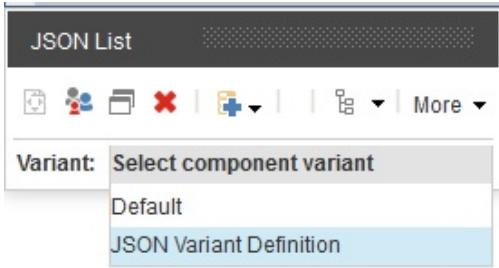
4. For the Image field, in the Variant details section, decide whether you want the image to display in Guid, Path, Raw, or Rendered mode.

The screenshot shows the 'Variant details' section for an 'Image' field. On the left, there's a sidebar with options: 'JSON List', 'Default', 'JSON Variant Definition' (which is expanded, showing 'Title', 'Content', 'Date', and 'Image'), 'JSON Results', 'Link List', 'Media Link', and 'Navigation'. In the main area, it says 'Item owner: sitecore\Adr'. Below that is another 'Variant details' section with a 'Mode [standard value]:' dropdown. The options are 'Path' (highlighted in blue), 'Guid', 'Raw', and 'Rendered'. The 'Path' option is currently selected.

Select a JSON variant

To select a JSON variant:

1. In the Experience Editor, navigate to the JSON rendering on the page, for example, the JSON List rendering, and in the Variant field click the variant from the drop-down list.



2. To view the new list, save the page.

```
"data" : [
    {
        "title" : "Creative Exchange",
        "content" : "Creative Exchange enables teams to work together on a single shared workspace. It allows users to share files, comments, and tasks in real-time, making it easier to collaborate on projects. Creative Exchange also includes a built-in versioning system that makes it easy to track changes and revert to previous versions if needed.",
        "date" : "04/03/2017",
        "image" : "/sitecore/shell/-/media/Creative-Exchange.png"
    },
    {
        "title" : "JSON",
        "content" : "The SXA JSON (JavaScript Object Notation) device renderer is a powerful tool for creating dynamic content. It allows you to define different content structures for different devices (e.g. desktop, mobile, tablet) and then automatically generate the appropriate markup based on the user's device. This makes it easy to create responsive websites that look great on all devices without having to write separate code for each one.",
        "date" : "04/18/2017",
        "image" : "/sitecore/shell/-/media/Default-Website/cover.jpg"
    }
]
```

3. To view the JSON variant, on the ribbon, on the Experience tab, click Other, and click Preview.

JSON Raw Data Headers

Save Copy

```
▼ data:
  ▼ 0:
    title: "Creative Exchange"
    ▶ content: "Creative Exchange enables...theme of the site.&nbsp;"
    date: "04/03/2017"
    image: "/-/media/Creative-Exchange.png"
  ▼ 1:
    title: "JSON"
    ▶ content: "The SXA JSON &nbsp; (Java...XA in JSON format.&nbsp;)"
    date: "04/18/2017"
    image: "/-/media/Default-Website/cover.jpg"
  ▼ 2:
    title: ""
    content: ""
    date: ""
    image: ""
  ▼ 3:
    title: "Toolbox"
    ▶ content: "Use the SXA Toolbox in th...edia plugins, and so on."
    date: "04/28/2017"
    image: "/-/media/SXA_tool-box.png"
```

The JSON search parameters

Abstract

Parameters to use for JSON data search calls.

You can view and access the JSON data that is returned to every page through the URL. In Preview mode, you can do JSON data search calls by adding query parameters in your browser's address bar. Each query starts with question mark (?) and each query parameter is separated by the ampersand (&) character. Values are comma separated.

Use the following parameters to search the JSON data:

Query parameter	Description
q	Returns the results with the specified word. For example, to search for pages with the word Creative: <code>&q=Creative</code>
p	Returns a specified amount of pages. For example, to return only the first 3 pages: <code>&p=3</code>
e	Excludes a specified amount of elements from the returned result. For example, to skip the first 4 elements of a page:

&e=4

Sorts the results based on specified facets.

o

For example, to sort results based on the title:

&o=Title, Ascending

Use the SXA Layout Service to model your pages in JSON

Abstract

Return placeholders and data source items in JSON.

SXA enables you to model your data in JSON (JavaScript Object Notation). For example, if you want to build a mobile app and feed it with SXA content, you can edit the JSON content on the page and have output content in JSON instead of HTML. You can use special renderings to build JSON layout but if your website uses default renderings, you can use the SXA Layout Service to return placeholders and data source items in JSON.

The SXA Layout Service uses serializers to render the JSON content. There is a generic serializer for all renderings that use variants and there are specific serializers for some of the renderings. You can view the settings in the Layout Service section of your rendering. For example, there is a Navigation serializer for the Navigation rendering:



This topic describes how to:

- [View your pages in JSON](#)
- [Filter JSON data using search parameters](#)

[View your pages in JSON](#)

To switch to the JSON device:

1. In the Experience Editor, on the ribbon, on the Experience tab, click Default, and then click JSON.
2. To view the JSON, on the ribbon, on the Experience tab, click Other, and click Preview.

The flat list in JSON represents the tree structure of your page. For example, if your page contains the page list rendering:

```
},
"components": [
  {
    "name": "Page List",
    "path": "main",
    "contents": [
      {
        "title": {
          "value": "Creative Exchange",
          "link": "/creative-exchange",
          "guid": "{D1F619A0-9C3B-43F4-B837-6E58D1A1EB17}"
        }
      },
      {
        "title": {
          "value": "Drag and Drop",
          "link": "/drag-and-drop",
          "guid": "{D8AFCAD4-2A8A-43CD-B9C5-165F65229852}"
        }
      },
      {
        "title": {
          "value": "Features",
          "link": "/features",
          "guid": "{BD0AC23B-76AD-412E-AAA2-552AD12B1DC8}"
        }
      },
      {
        "title": {
          "value": "Page",
          "link": "/page",
          "guid": "{38C345EF-B8CE-4506-A7DF-BCDA4E1DF2EE}"
        }
      }
    ]
  }
]
```

Or an image:

```
        },
    "components": [
        {
            "name": "Image",
            "path": "main",
            "contents": {
                "image": {
                    "fieldType": "image",
                    "link": "http://sxa36/-/media/Project/Test/Variants/Gepard-Animals.jpg",
                    "guid": "{BB981B56-CF6E-4F1C-B710-FC3A1AB60A40}"
                },
                "imagecaption": "Example image"
            }
        }
    ]
}
```

Note

If you applied personalization rules to your webpage, these are also displayed in JSON. For example, if you add a rule to hide a rendering under certain conditions.

Filter JSON data using search parameters

You can view and access the JSON data that is returned to every page through the URL.

To filter JSON data using search parameters:

- In Preview mode, to make JSON data search calls, add query parameters in your browser's address bar. You must start each query with a question mark (?) and separate each query parameter with the ampersand (&) character. Values are comma separated.

You can filter your search by component type (`sc_filter_name`), placeholder (`sc_filter_path`), and component ID (`sc_filter_id`).

For example:

Query

= /home?sc_device=json&sc_filter_path=/leftph/menu

= /home?sc_device=json&sc_filter_id=content,spot1

= /home?sc_device=json&sc_filter_name=richtext

= /home?sc_device=json&sc_filter_path=content&sc_filter_name=rich

Trigger a JSON layout rendering

Abstract

SXA enables you to model your data in JSON (JavaScript Object Notation). For example, if you want to build a mobile app and feed it with SXA content, you can edit the JSON content on the page and have output content in JSON instead of HTML.

To trigger JSON output, you can:

- Trigger JSON output using `device_id`. You can include the device id in the URL. Use `sc_device` to change the device in which the website is displayed. You can use the device name or the device id. For example, if you have a JSON layout defined for this page:

```
http://sxa18//json-content
```

Modify it by adding the `sc_device` query string:

```
http://sxa18//json-content?sc_device={0B4A31C9-712D-4578-A396-2DDC0F34B63A}
```

or

```
http://sxa18/json-content?sc_device=json
```

- Trigger a device change with HTTP headers using `HTTP_header`. Set the `application/json` value for the `ContentType` header. For example, invoke with Powershell:

```
Invoke-WebRequest "http://sxa18//json-content" -ContentType "application/json"
```

Best practices

The recommended practices described in this section can help you work with SXA more efficiently:

- [Recommendations: Working with data sources and media](#)
- [Recommendations: Extending SXA](#)
- [Recommendations: Sharing content](#)
- [Recommendations: Structuring pages](#)
- [Recommendations: Enhancing SXA performance](#)
- [Recommendations: Working with themes](#)
- [Recommendations: Using renderings](#)

Recommendations: Working with data sources and media

Abstract

Recommended practices for structuring and naming of data sources and media items

Organizing and naming your data sources and media items wisely can make a world of difference for your editors. This topic describes some best practices for dealing with data sources and media items in SXA.

- [Clean up unused data sources](#)
- [Don't put media items directly under the site's Media folder](#)
- [Give data sources meaningful names](#)
- [Organize site data sources in folders](#)
- [Run source field reports to help set the data source context](#)

[Clean up unused data sources](#)

Remove unused site data sources. Keeping unused site data sources in the site introduces the following risks:

- Your editors can use them by accident.
- Your editors might not be able to find a more suitable piece of content that you intend them to use.

Note

This recommendation applies to the site/global/shared data sources and is not necessary for the data sources stored under the *Data* folder under your page.

Do not put media items directly under the site's Media folder

The *Media* folder in SXA sites (`sitecore/Content/Tenant/Site/Media`) is a scoped view of the Media Library items available for the site. You must not upload Media items to this folder. Instead, use the general *Media Library* folder (`sitecore/Media Library`).

Note

From SXA version 1.7 and higher, you cannot upload media items under the *Media* item of your site.

Give site data sources meaningful names

Always use meaningful names for your data sources that you use across sites. Think of a name that indicates its purpose so that editors can identify and reuse them. A name such as *Carousel 1* does not encourage reusability. However, *Hero Carousel for Christmas 2018* does.

Note

This recommendation applies to the site/global/shared data sources and is not necessary for the data sources stored under the *Data* folder under your pages.

Organize site data sources in folders

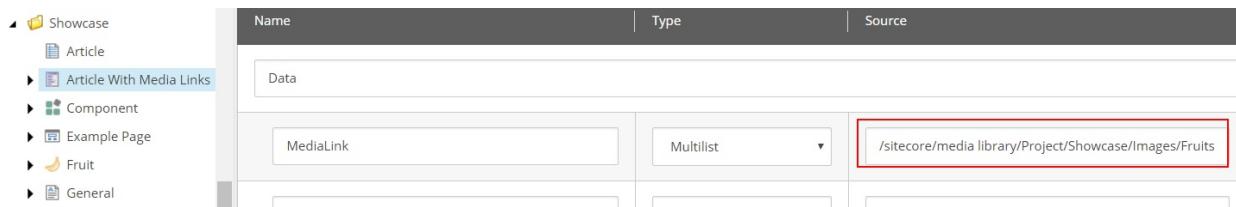
Site data folders are expected to host reusable content and it is best practice to keep them uncluttered. Group them in subfolders as needed so that your editors can easily locate and repurpose content that is meant to be reusable.

Note

This recommendation applies to the site/global/shared data sources and is not necessary for the data sources stored under the Data folder under your page.

Run source field reports to help set the data source context

Most SXA renderings are designed for reusability and to pull data from data source items. This means that the content they display is not bound to the page on which they appear but is stored in data source items. The Source field specifies where the user is allowed to look for the data source.



Setting this location correctly makes selecting data sources much easier because, for example, the Content Editor is directed to a specific site subfolder and does not have to search the whole Media Library for an image. You can set the location manually, use a query, or use the [Source field report](#) helper script.

Recommendations: Extending SXA

Abstract

Recommendations for extending SXA.

You might want to [extend SXA](#), for example, to add your own custom rendering to the SXA toolbox, add a theme, or a template. This topic describes some requirements and recommendations for extending SXA.

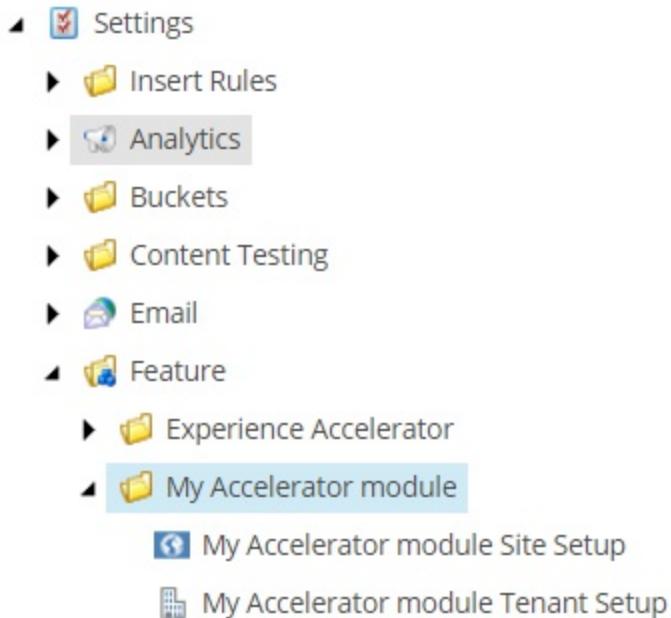
- [Never put custom items in SXA-controlled branches of the tree](#)
- [Do not modify OOTB SXA items](#)
- [Do not replace the SXA MVC layout](#)
- [Create an SXA module for your custom renderings](#)
- [Follow Helix principles when adding functionality to your solution](#)
- [Consider using existing renderings before building a new one](#)
- [Consider cloning existing renderings before building a new one](#)
- [Limit scope of fields linking to items](#)

Never put custom items in SXA-controlled branches of the tree

When [extending SXA](#), think about where to store your custom items. SXA overwrites the standard SXA sections during SXA updates. Therefore, placing custom items in tree branches controlled by SXA, can result in your instance being broken after an upgrade. We recommend that you:

- Place [custom modules](#) in a branch that is a sibling to the *Experience*

Accelerator module root.



- Place [templates](#) in folders that are siblings to the *Experience Accelerator* folder – never inside.
- Similarly, you must create a module-specific root item for all other areas, such as Renderings, Branches, Media.

Related topics:

- [Extending SXA](#)
- [Add and extend an SXA template](#)
- [Create a new SXA module](#)

Do not modify OOTB SXA items

When [extending SXA](#), always use the SXA extension points for custom functionality. For example, [create a custom data template](#). If this is not possible and you have a good reason for the extension, contact the SXA team to create extension point for you.

Related topics:

- [Extending SXA](#)
- [Add and extend an SXA template](#)

Do not replace the SXA MVC layout

The SXA MVC layout is the basis of grids, themes, and meta components, and SXA can depend on its presence for other common behavior. If you replace the layout with your custom implementation, you risk losing compatibility with future versions of SXA.

If you need to add custom non-visual elements to the layout, consider creating a meta rendering and add it to the [Meta Partial Design](#). If you have visual elements, consider using standard SXA renderings.

SXA provides multiple possibilities to extend the layout. For example, you can:

- Add non-visual extensions by using meta-partial designs.
- Add visual extensions by using regular partial designs.
- Add JavaScript and styling elements using the assetService[pipeline](#).
- Define a [custom grid](#) and define your custom markup of the <body> tag.

Create an SXA module for your custom renderings

Place custom renderings in an SXA [module](#) that can be added to an existing site or used immediately in newly created sites.

You must:

- Provide [site/tenant scaffolding integration](#) for your modules.
- Include data source definitions for your components (or re-use OOTB

SXA data sources).

- Define site-relative queries for your data sources.

Related topics:

- [Extending SXA](#)
- [Add and extend an SXA template](#)
- [Create a new SXA module](#)
- [Add modules to Site and Tenant scaffolding](#)

Follow Helix principles when adding functionality to your solution

Helix offers a set of official guidelines and recommended practices for Sitecore development. SXA follows the Sitecore Helix principles that contain development process recommendations for building, testing, [extending](#), and maintaining Sitecore implementations.

We recommend that you:

- Do not add dependencies on SXA feature modules – but rather only on foundation modules. It is permitted to violate the rule while cloning renderings because you use the platform code to provide the new feature. This recommendation applies to the OOTB SXA renderings and not necessarily SXA Storefront renderings.
- Separate and divide your features into proper Helix modules.
- Provide [site/tenant scaffolding integration](#) for your modules.

Related topics:

- [Extending SXA](#)

- [Add and extend an SXA template](#)
- [Create a new SXA module](#)
- [Add modules to Site and Tenant scaffolding](#)

Consider using existing renderings before building a new one

You can customize existing SXA renderings by using rendering variants and composites. If possible, always consider using variants and composites before building a custom rendering.

For example, with rendering variants you can: use a [reference item](#) to display fields, use a [query item with a token](#) to render context items

Related topics:

- [The SXA renderings and rendering variants](#)
- [Create a rendering variant](#)
- [Using rendering variants to display product information](#)

Consider cloning existing renderings before building a new one

You can easily [clone SXA renderings](#) if you want to customize the HTML beyond what is possible using rendering variants. In the process, you can also add to their data sources and rendering parameters. If possible, always consider this approach before building a custom rendering for functionality that you can achieve with SXA OOTB components.

Note

This approach is available since SXA 1.6.

Limit scope of fields linking to items

For fields that support queries, you must limit the scope of items that editors can choose from.

For example, use:

- `query:$siteMedia` – for file or image fields.
- `query:$home` – for fields that link to pages.
- `query:$site` – for fields that can link to any item within your site.

The description of the [SXA pipelines](#) includes a complete list of tokens and how to extend them.

Recommendations: Sharing content

Abstract

Control content from one central location.

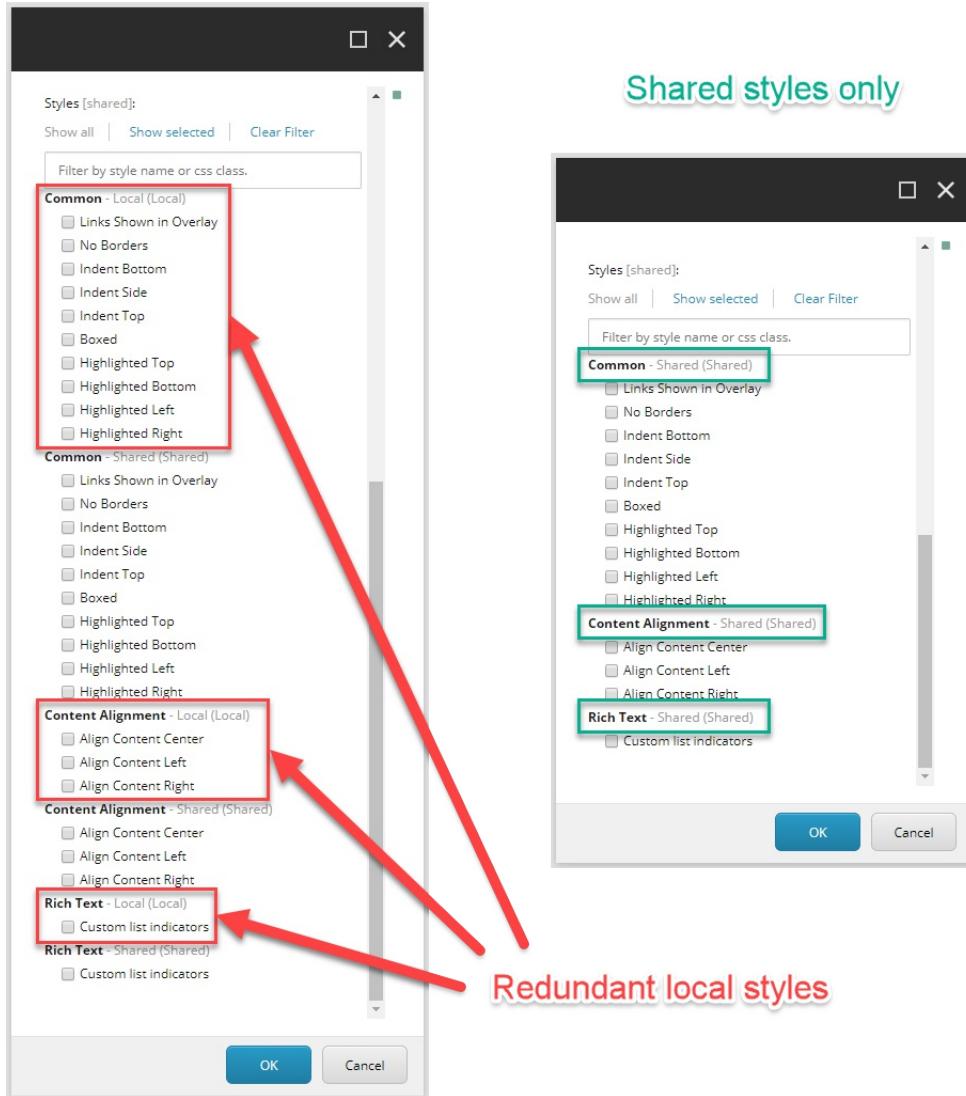
SXA lets you share styles, data sources, page designs, partial designs, and rendering variants between the sites within one tenant.

This topic describes the following recommendations for sharing content with SXA:

- [Consider using a shared site as the exclusive style container in a tenant](#)
- [Consider defining your rendering variants in the shared site](#)
- [Consider defining your page designs and partial designs in the shared site](#)
- [Consider using delegated areas for pages that share content across sites](#)
- [Consider creating a master site to clone for a rollout to new markets](#)

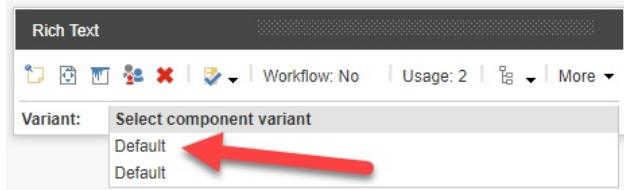
Consider using a shared site as the exclusive style container in a tenant

When you decide to [mark a site in a tenant as shared](#), it contributes its styles to all other sites in that tenant. Consider removing redundant styles and use the shared site's styles instead. Otherwise, your editor is presented with duplicated styles.



Consider defining your rendering variants in the shared site

SXA lets you [share](#) styles, data sources, page designs, partial designs, and rendering variants between the sites within one tenant. When you decide to [mark a site in a tenant as shared](#), all its rendering variants are made available for all other sites in that tenant. Consider using these rendering variants and remove the redundant versions from other sites. Otherwise, you may end up with duplicated variants presented to your editor.



Consider defining your page designs and partial designs in the shared site

When you decide to [mark a site in a tenant as shared](#), all its Page Designs and Partial Designs are made available for all other sites in that tenant. Consider designing them once in the shared site and re-using them in other sites. This practice allows you to manage the designs centrally and automatically publish designs across all sites using them.

Consider using delegated areas for pages that share content across sites

As well as [sharing presentation and data sources](#) between sites in the same tenant, you can also share pages. [Sharing content as a delegated area](#) lets you control the content from one central location in the content tree. Whenever you have a set of pages that you want to centrally manage but then publish across multiple sites – delegated areas are the way to go. For example, if you want to reuse blog posts from your master site and do not want to have to add every new blog post manually.

If you edit a field in the original item and save it, SXA updates the field in all the cloned items in the delegated area. If you enable the use of the SXA delegated area when you clone items from a master site to a local site, the cloned items are automatically updated and published whenever the original items are.

Consider creating a master site to clone for a rollout to new markets

To speed up rolling out the site you are creating to new regions and users, consider creating a master site that has all the basic functionality and stub content required by new projects. After the site creation, leverage the [SXA's site cloning](#) functionality to speed up the process of the rollout for new users.

Packaging tenants with the SXA Tenant Exporter is the easiest way to [move tenants between environments](#).

Recommendations: Structuring pages

Abstract

Best practices for layout

In SXA, you build your website with reusable pieces of content and layout. All these pieces together create the layout for your pages.

This topic describes the following recommendations for working with [page designs](#), partial designs, snippets, and composites:

- [Design the page layout in partial designs](#)
- [Put complex reusable structures in partial designs or composites](#)
- [Remove unnecessary renderings from your site](#)
- [Set placeholder restrictions](#)
- [Do not use Standard Values to set presentation details for your pages](#)
- [Use partial design inheritance](#)
- [Prioritize changing the grid settings of a rendering over using splitters to change the layout of a rendering](#)

[Design the page layout in partial designs](#)

To make the editor's life easier, design as much as possible using [partial designs](#). Partial designs help create a consistent style, make the editor's work easier, and improve Experience Editor performance.

What you should know about partial designs:

- When renderings are placed on partial designs and assigned to a page through [page designs](#), editors need to perform fewer operations when creating generic pages, such as articles.
- Place an empty Container rendering or another rendering with an empty placeholder on your partial design to give your editors well-defined extension points. Set [placeholder restrictions](#) on those placeholders to indicate which components are allowed in those areas.
- Leverage [Snippets](#) and page branches where local customization is needed. Partial designs improve performance.

Related topics

- [Page designs](#)
- [Create and change a partial design](#)
- [Create and assign a page design in the Content Editor](#)
- [Create and assign a page design in the Experience Editor](#)

Put complex reusable structures in partial designs or composites

Do not make editors place multiple renderings to create a page that does not need a unique layout. For example, all article pages tend to look the same, which makes them a perfect candidate to define their structure in [partial designs](#). In contrast, landing pages tend to vary a great deal, which makes them good candidates to have a minimal page design with large placeholders giving editors a great deal of flexibility.

Use partial designs when the page layout should be determined ahead of page creation.

If you want the editors to be able to style and change the layout, consider using composites or snippets.

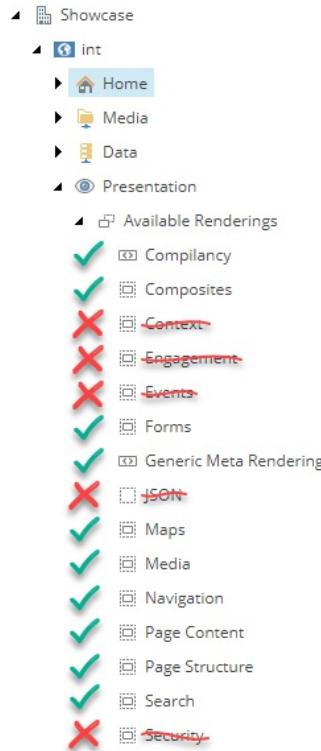
Use [composites](#) when you want a group of renderings to have its own layout and you want the editor to be able to design it in the Experience Editor. Consider using [snippets](#) when you want to create a reusable group of renderings. A snippet consists of several renderings that can be designed separately in the Experience Editor. In the same way as with partial designs, you can group different kinds of renderings and reuse this group in different locations. The difference between a snippet and a partial design is that snippets are extensible, you can change the styling, and you can use them as often on the page as you want.

Related topics:

- [Page designs](#)
- [Create and change a partial design](#)
- [Create and assign a page design in the Experience Editor](#)
- [Share datasources and presentation using the multisite feature](#)

[Remove unnecessary renderings from your site](#)

If you are not using some renderings and you do not have them styled – consider removing them from the Available Renderings section in the site. A clean toolbox, makes the editor's life a lot easier. It can be frustrating to drop a rendering on a page only to find out that it was not styled and does not fit the branding of your website.



Set placeholder restrictions

To have some control over the renderings that can be placed in a placeholder, you can [set placeholder restrictions](#). Excluding renderings that you never expect to use, or do not want to use, makes the editor's job much easier.

Related topics

- [The toolbox](#)
- [Set placeholder restrictions](#)
- [Restructure the toolbox](#)

Do not use Standard Values to set presentation details for your pages

All sites in a single Tenant use the same set of templates. Each of them can have separate style items or even use a different grid. Unless you are using a

single [shared site](#) for the whole tenant as your styles' holder, this can result in broken layouts on some sites.

As an alternative, use partial designs or [snippets](#) with sets of preconfigured renderings. Setting your presentation up in this way creates a balance of flexibility, and makes the experience more streamlined and manageable.

If you have designs that you want to share between all sites within a tenant, consider creating [shared site functionality](#) that contains your page and partial designs so that you can reuse those.

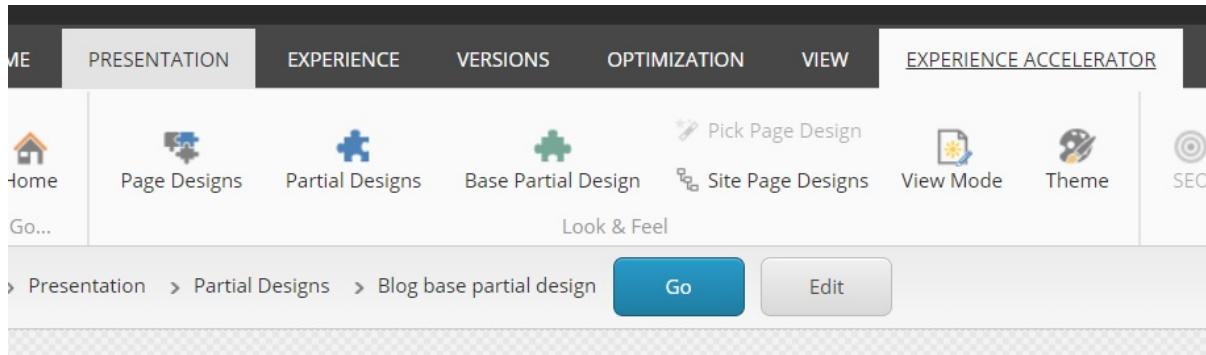
Related topics:

- [Page designs](#)
- [Create and change a partial design](#)
- [Create and assign a page design in the Content Editor](#)
- [Create and assign a page design in the Experience Editor](#)

Use partial design inheritance

When you create partial designs that only differ from each other slightly, consider creating a base [partial design](#) with the common layout elements and then inherit that partial design in the more specific partials. With partials' inheritance, you can maintain the shared aspects of your design in one place and easily propagate those changes to all the different page types that the authors are creating.

For example, if you need two slightly different headers. By taking advantage of partials' inheritance, you can create one base partial design with, for example, the navigation rendering, and then create other partials on top of that one and fill in the differences with different logos, and so on. This way, you avoid the creation of numerous templates with only minor differences.



Prioritize changing the grid settings of a rendering over using splitters to change the layout of a rendering

By default, when you add a rendering to a page, it takes the full width of the column. In the Experience Editor, you can change this in [the grid settings of the rendering](#). To change the default settings permanently, you must change the grid settings in the Content Editor.

Always consider changing the layout of a rendering first before placing additional splitters on a page. This way, you limit the number of renderings, which makes your page more responsive, and improves the speed of the Experience Editor.

Related topics:

- [The grid layout](#)
- [Change the layout of a rendering](#)
- [The grid settings](#)
- [Change the layout of a page](#)

Recommendations: Enhancing SXA performance

Abstract

Enhance SXA performance

This topic describes the following recommendations to improve the load time of your pages:

- [Set HTML caching settings for renderings](#)
- [Limit the number of renderings on a page](#)
- [Disable content testing for editors who do not require that functionality](#)
- [Verify that you have enabled the Asset Optimizer](#)

[**Set HTML caching settings for renderings**](#)

Consider configuring the rendering's HTML caching settings to improve the visitor's experience. SXA provides a rich and [configurable HTML caching layer](#) including donut caching for renderings. With SXA, you can cache the whole of your page apart from a single rendering or two that you want to personalize.

[**Limit the number of renderings on a page**](#)

Do not use too many renderings on a page; we recommend absolutely no more than 30. Having too many editable renderings on a single page can slow down the Experience Editor. Consider putting all [common elements used across multiple pages](#) in [partial designs](#).

Disable content testing for editors who do not require that functionality

Content testing uses scheduled tasks for different actions, such as rebuilding testing indexes. If you do not use this feature, [disable content testing](#) to avoid unnecessary server load. Content testing is a major contributor to the time the Experience Editor requires to render a page for editing.

During the site development or if you do not plan to use content testing, switch off this functionality using a config patch. Do not place users who are not expected to start tests in the Analytics Testing role. The Experience Editor does not load the Content Testing UI for those users.

Verify that you have enabled the Asset Optimizer

The [SXA Asset Optimizer](#) improves the end-user experience by optimizing CSS styles and JavaScript, and reducing the amount of data that needs to be transferred. Ensure that the [Asset Optimizer is enabled for your site](#) to limit the number of round trips to the serve in a production environment.

Recommendations: Working with themes

Abstract

Organize classes and assets for your themes.

This topic describes the following best practices for extending your themes:

- [Do not modify platform themes](#)
- [Assign custom styles to renderings](#)
- [Clean up your styles folder](#)
- [Organize custom styles in folders](#)

[Do not modify platform themes](#)

Make a copy, modify your copy, and then inherit from the copied theme instead of from the original one. This practice is merely a reinforcement of the following practices:

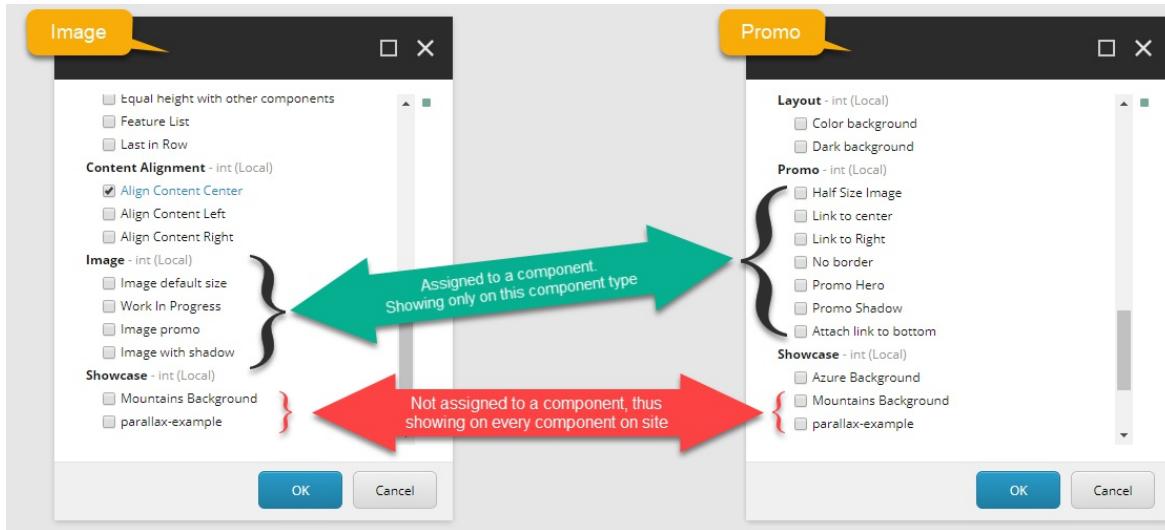
- Never put any custom items in SXA controlled branches of the tree.
- Do not modify the OOBT SXA items.

If you do not follow this recommendation while [extending SXA](#), you risk getting your changes overwritten the next time you upgrade SXA.

[Assign custom styles to renderings](#)

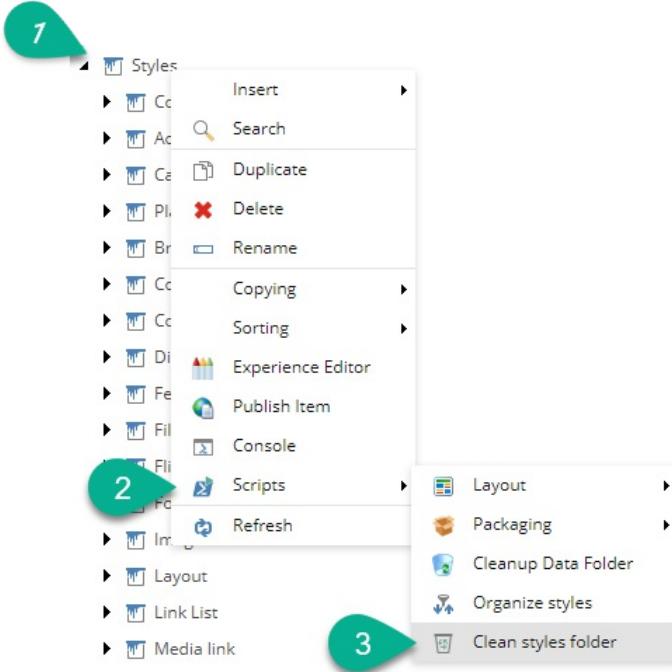
If you [create a custom style for a rendering](#), make sure you assign it to the correct renderings.

For example, if you provide a style that is specific to your Promo rendering but you do not want other renderings to use that style, you can achieve this by listing renderings that the style applies to in the style item, in the **Allowed Renderings** field. If you do not assign the style to any rendering, it is available for every rendering.



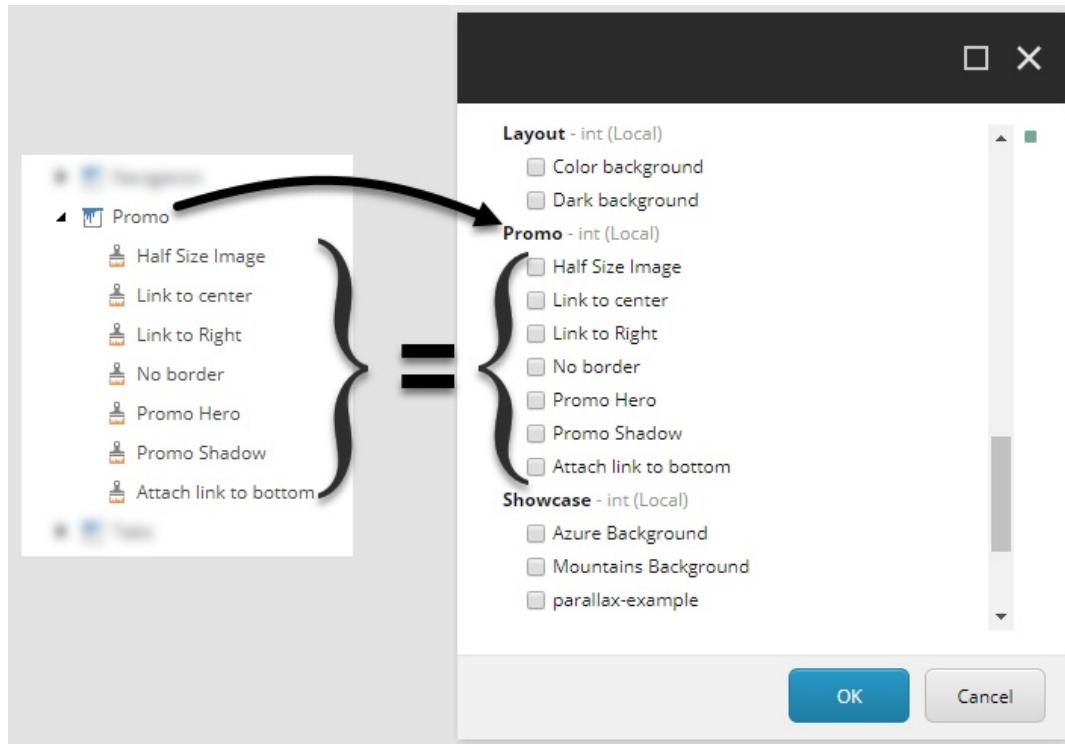
Clean up your styles folder

Remove unused styles from the styles folder. It can be frustrating for a editor of your site to apply a style only to find out that it made no change to the style of their rendering. SXA includes a number of PowerShell [scripts](#) to automate the most common tasks. Right-click the **Styles** item, and then click **Scripts** to find a clean-up script.

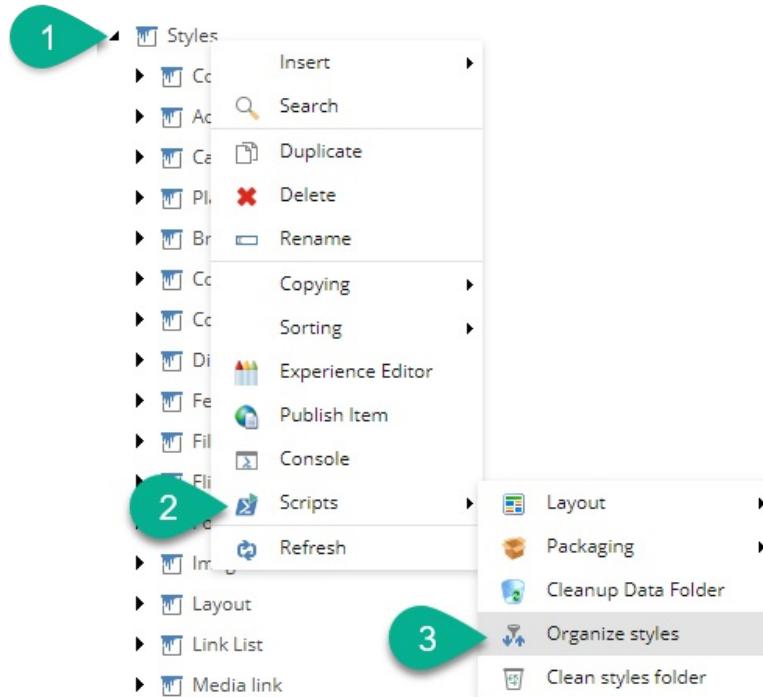


Organize custom style items in subfolders

As you deliver the site branding and theme, you can expect your site styles list to grow rapidly. Consider organizing the styles into subfolders to avoid overwhelming your editors with a long list of uncategorized styles.



SXA includes a number of PowerShell [scripts](#) to automate the most common tasks. Right-click the *Styles* item, and then click *Scripts* to find the *Organize styles* script.



Recommendations: Using renderings

Abstract

Think about what rendering to use and how to use it.

SXA comes with a library of predefined renderings and rendering variants to ensure modular component based design. You can also create a custom rendering (variant). Using renderings wisely can help you get work done more efficiently.

This topic describes the following recommendations:

- [Use the proper rendering for the job](#)
- [Use the Snippet rendering for a group of renderings](#)
- [Do not use the Plain HTML rendering for content that is edited by authors](#)
- [Make sure the Rich Text field content is fully editable](#)

[Use the proper rendering for the job](#)

Avoid using versatile renderings such as Page Content, Page List, Promo, or Rich Text to render most of your page layout. Use the correct rendering for the purpose. Consider [cloning](#) the standard rendering and give it a name that describes the rendering's purpose.

For example, the names of the following clones of default renderings make it easy for business users to understand their purpose instantly:

- **Teaser** (a clone of the Page Content rendering) that is configured to

link to pages to tease their content.

- **Highlight or Call to Action** (a simple copy of the Promo rendering) to provide a set of styles and rendering variants fit for the job.
- **Hero Carousel** (a clone of the Carousel component) with preconfigured slide data sources and components with rendering variants specific for your project.

Use the Snippet rendering for a group of renderings

If your editors often use a set of renderings to form a section of a page, consider using the Snippet rendering.

The [snippet](#) rendering supports embedding multiple renderings, allows you to [configure](#) them, and allows you to specify that a copy of the snippet, its configuration, and content are added to the page. When editors change the snippet, the changes are only applied for that specific page.

Do not use the Plain HTML rendering for content that is edited by authors

For complex components, you might consider using the Plain HTML rendering because it provides complete freedom regarding the HTML you can place in it. However, this way of adding content to a page can be difficult for your editors to work with because not all content editors know how to edit the HTML code. Consider using clones, [variants](#), or the HTML snippet instead of the Plain HTML rendering.

Before using the Plain HTML rendering, consider these other options:

- [Copy and customize](#) an SXA rendering (such as the Promo rendering).
- Create a more elaborate rendering variant. This way, you can take full control over the component markup (in the rendering variant) while still making the text and images editable for your editor. This is the preferred

approach for visual components.

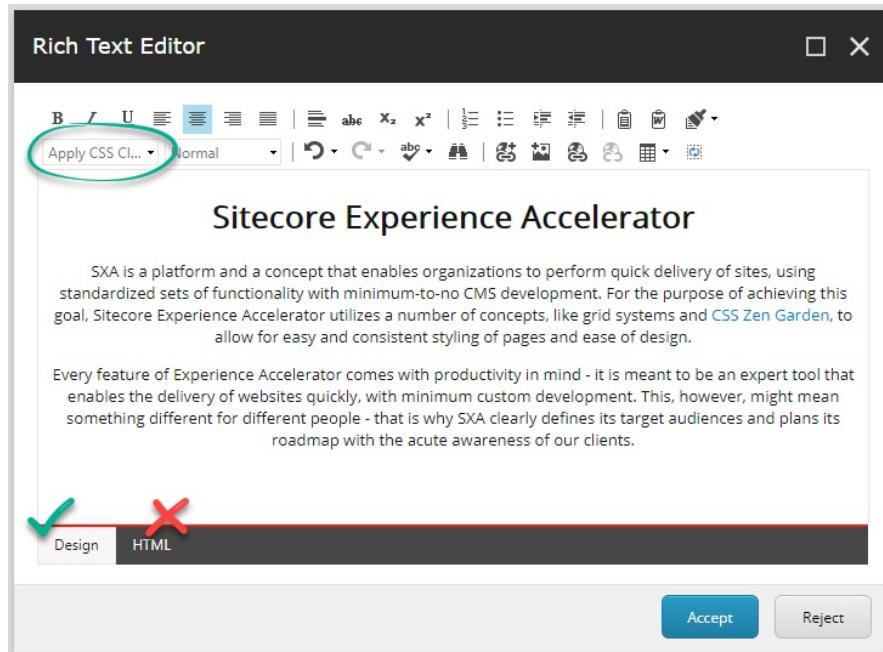
- Use an HTML snippet on your Meta Partial Design. If a component has no visual content to be edited (for example, because you want to place JavaScript code on your page), consider using an HTML Snippet on your Meta Partial Design.

Note

This functionality is available in SXA version 1.7 and higher.

Make sure the Rich Text field content is fully editable

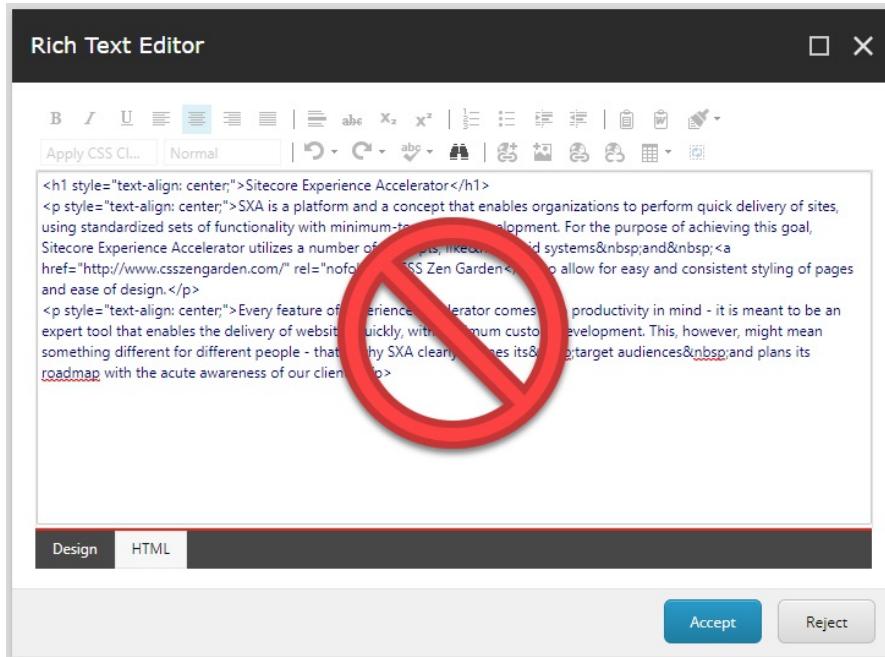
When providing texts that marketers are expected to maintain after the site is deployed you must **never** put in any markup or classes that cannot be maintained using the WYSIWYG editor.



If editors need to put classes on text elements (that you need to style in a specific way in your theme) you must add these classes to the Sitecore **Rich Text Editor**. In this way, they are available in the drop-down menu. Editors

can then style the content without having to switch to the HTML view.

Also, you must make sure that the style is reflected in the appearance within the Rich Text Editor, to avoid surprises after the content publication.



If your pages need more elaborate HTML code, consider [cloning a component](#), configuring a rendering variant, or [creating a custom component](#), rather than placing complex HTML in the Rich Text field.

Sitecore Experience Accelerator (SXA) FAQ

Commonly asked questions on SXA installations:

- [Can you mix SXA and non-SXA websites within a single Sitecore instance?](#)
- [Can you host a non-SXA site within an SXA tenant?](#)
- [Can SXA pages contain non-SXA components?](#)
- [Can I use SXA components outside of an SXA site?](#)
- [Is there a limit on the number of sites that you can have in a single SXA installation?](#)
- [Is the Sitecore PowerShell Extensions \(SPE\) required to be installed on every Instance in the environment?](#)
- [Where can I download the SPE WDP files required for Azure deployment?](#)
- [What is Sitecore doing to prevent components from breaking between versions?](#)

Can you mix SXA and non-SXA websites within a single Sitecore instance?

Yes (and no). The challenges of mixing SXA with other solutions are the same as with running any two solutions from different implementers on a single Sitecore instance. Non-SXA sites and SXA sites share pipelines, and both attempt to add processors and modify Sitecore behavior to its needs.

SXA did extensive reviews of all pipelines within recent releases. We have made sure we abort our processors whenever we detect that they are running outside of the context of SXA sites, for the very purpose of limiting any interference that SXA could cause for another solution.

We have tested it working with the Habitat and confirmed that they could run side by side. SXA should be as unobtrusive as any Sitecore solution can be to another solution.

On the other hand, SXA is not in control of the behavior of the processors from the other solution - which means the cohabiting code also needs to be a good neighbor and check for interference with SXA's operation if it is to be deployed to the instance that co-hosts SXA.

Important

Integration due diligence QA is always necessary when there are two solutions running in a single process.

Can you host a non-SXA site within an SXA tenant?

Yes. A non-SXA site does not participate in a tenant's operations, but it should not cause problems either. For SXA, a tenant is mostly a container for templates and the grouping under /sitecore/content is only reflecting that. It is the site that is a unit of execution.

Can SXA pages contain non-SXA components?

Yes. SXA pages can contain both SXA and non-SXA components. You can include these components in sitecore/Content/Tenant/Site/Presentation/Available renderings. The components can be placed on the page and will function as long as they are not making assumptions as to the site structure that is conflicting with SXA. If this happens, you can usually mitigate this fairly easily.

You can choose to create an SXA rendering of your component. If you

choose not to convert the component into SXA components, the component will not be able to:

- Use SXA styles. Styling these components can be more difficult.
- Use SXA grid settings. Resizing these components can be more difficult.
- Participate in the Creative Exchange styling process.

Can I use SXA components outside of an SXA site?

No. SXA components do not function outside of SXA sites. Because SXA components make many assumptions about, for example, the site structure, the data sources, this possibility is disabled.

Is there a limit on the number of sites that you can have in a single SXA installation?

No. SXA has no inherent limit other than those of the platform itself. SXA is as scalable as the Sitecore environment to which you deploy it. Please consult the general platform architecture recommendations with your partner.

Is the Sitecore PowerShell Extensions (SPE) required to be installed on every instance in the environment?

No. SPE module is only used or required for certain Content Management tasks, and you should not deploy it on Content Delivery servers.

Where can I download the SPE WDP files required for Azure deployment?

There is a matching SPE WDP file on the download page in the Azure

section for every SXA release.

What is Sitecore doing to prevent components from breaking between versions?

- Most of the components are rendered using Rendering Variants that are defined as Sitecore items and are in full control of the client/developers.
- The JavaScript that provides functionality for a component exists in a Base theme that is under SXA's control. Extracting the JavaScript from the site theme means that when we find a problem or need to provide additional functionality, Sitecore can do it without interfering with the client theme.
- The markup of components does not change often.
- SXA 1.8 introduced the ability to replace markup for existing components, which means that even if we changed it in a future version, you could revert it to the old markup if needed.
- When SXA introduces a big change to a component, it is marking the old component as obsolete (and does not deploy it to sites that you create after the upgrade). Your existing sites can keep using the old version.
- When the data structures for components change (data source or rendering parameters), we provide migration scripts that perform the transformation to a new structure.

Important

You must always verify that your website looks and feels consistent with how it worked before the upgrade. Themes are considered customization and Sitecore does not modify your themes to comply with any possible changes introduced to the markup.

Table of Contents

[Setting up and configuring](#)

[Structure](#)

[The SXA items in the Content Editor](#)

[The tenant items](#)

[The site items](#)

[The custom items](#)

[Layout](#)

[Set placeholder restrictions](#)

[Set placeholder restrictions in the Content Editor](#)

[Set placeholder restrictions in the Experience Editor](#)

[Merge placeholder restrictions](#)

[Use a query to determine the data source context](#)

[Add a query child item to a rendering variant](#)

[Use a query in the data source field of a rendering](#)

[Run a source field report to set the data source context](#)

[Performance](#)

[Set SXA caching options](#)

[Set rendering definition caching options globally](#)

[Set site level rendering caching](#)

[Set placeholder caching](#)

[Set specific rendering caching options](#)

[Rendering variants](#)

[Create a rendering variant](#)

[Render image fields using the Responsive Image variant](#)

[Render an image field](#)

[Render multiple images in a link field](#)

[Enable field fallback functionality](#)

[Walkthrough: Creating a rendering variant for a blog post](#)

[Add the template](#)

[Create the variant](#)

[Create the partial design](#)

[Create the page design](#)

[Assign the page design](#)

[Walkthrough: Using the rendering variant reference item to display fields from a linked item](#)

[The scenario and the end result](#)

[Create the rendering variant reference item](#)

[Render the fields](#)

[Render related media](#)

[Walkthrough: Using a query to render items](#)

[The scenario](#)

[Add the rendering variant](#)

[Add the reference item](#)

[Add the query item](#)

[Add a query item that uses a token](#)

[Search](#)

[Configure SXA indexing](#)

[Boost search results for SXA site queries](#)

[Localize date formats for SXA search](#)

[Integrating SXA with Sitecore Cortex™ Content Tagging](#)

[Security](#)

[Set up security for a tenant and a site](#)

[Set up security for a tenant](#)

[Set up security for a site](#)

[The SXA security roles](#)

[Use a custom SXA login page](#)

[Sites](#)

[Add and select a custom link provider](#)

[Add a language version to an SXA site](#)

[Add a style for a rendering](#)

[Clone an SXA site](#)

[Manage multiple sites with the SXA Site Manager](#)

[Configure a sitemap](#)

[Configure a snippet](#)

[Configure the map provider](#)

[Add the map base theme to a theme](#)

[Store the maps authorization key](#)

[Generate a custom static error page](#)

[Restructure the SXA Toolbox](#)

[Change the order of rendering categories](#)

[Change the order of renderings within a category](#)
[Add a custom category](#)
[Add a module to an existing tenant or site](#)
[Add modules to a specific tenant or site](#)
[Add modules to multiple tenants and sites using the Bulk module installer](#)
[Exclude a page from the navigation menu](#)
[Hide a page using a navigation filter](#)
[Create a custom navigation filter](#)
[Adjust link settings to enable cross-site linking](#)
[Configure the scope of the Media Library for an SXA site](#)
[Tenants](#)
[Create a tenant and a site](#)
[Add and extend an SXA template](#)
[Create an SXA template](#)
[Extend an SXA template](#)
[Export a tenant](#)
[Add a module to an existing tenant or site](#)
[Add modules to a specific tenant or site](#)
[Add modules to multiple tenants and sites using the Bulk module installer](#)
[Remove a tenant or a site](#)
[Extending SXA](#)
[Development](#)
[Add a dynamic placeholder to a rendering variant](#)
[Change or add database content](#)
[Add modules to Site and Tenant scaffolding](#)
[Add a module scaffolding definition](#)
[Add scaffolding actions](#)
[Build a rendering that includes variants](#)
[Controller](#)
[Repository](#)
[Model](#)
[View](#)
[Copy and customize a rendering](#)
[Add a new module](#)
[Copy and customize a rendering](#)
[Add a rendering to the toolbox](#)
[Create a new SXA module](#)

[Data sources](#)

[No data source item](#)

[Optional data source item](#)

[Single data source item per platform/tenant/site](#)

[Reusable data source](#)

[Select Page as a data source](#)

[Auto-generated data source](#)

[Run a source field report to set the data source context](#)

[The SXA pipelines](#)

[resolveVariantTokens](#)

[ioc](#)

[decoratePage](#)

[decorateRendering](#)

[mediaRequestHandler](#)

[resolveTokens](#)

[assetService](#)

[getControlEditability](#)

[getRobotsContent](#)

[getRenderingCssClasses](#)

[refreshHttpRoutes](#)

[getVelocityTemplateRenderers](#)

[resolveSearchQueryTokens](#)

[resolveBoostingQuery](#)

[The SXA script library](#)

[Walkthrough: Building a simple rendering](#)

[Create a rendering](#)

[Create the controller and action method](#)

[Inject a repository into a controller](#)

[Create a controller rendering in Sitecore](#)

[Register dependencies](#)

[Add grid and styling support](#)

[Add a rendering to the toolbox](#)

[Customize rendering HTML per site](#)

[Embed HTML in SXA](#)

[Add the Plain HTML component](#)

[Add an HTML Snippet](#)

[Create a custom token for a rendering variant](#)

[Add a data source setting for an existing rendering](#)

[Grid](#)

[Create a custom grid](#)

[Designing](#)

[The SXA themes](#)

[Assign a theme](#)

[Extend a theme](#)

[Create an extension for a theme](#)

[Extend a theme using the Extend Site Theme action](#)

[Extend a theme using the Attach Theme Extension action](#)

[Create a custom theme](#)

[The View Modes](#)

[Page designs](#)

[Create and assign a page design in the Content Editor](#)

[Create a Page Design](#)

[Assign a Page Design](#)

[Create and assign a page design in the Experience Editor](#)

[Create a page design](#)

[Assign a page design](#)

[Create and change a partial design](#)

[Front end](#)

[Working with Creative Exchange](#)

[Change a site design using Creative Exchange](#)

[Change an image](#)

[Change the layout](#)

[Change the styling of text renderings](#)

[Modify your site design with Creative Exchange Live](#)

[Style SXA sites with Sass](#)

[Trigger Creative Exchange from your CI server](#)

[Trigger export and import](#)

[Change export options using the Get-CreativeExchangeExportResponse cmdlet](#)

[Change import options using the Get-CreativeExchangeImportResponse cmdlet](#)

[Import and export a web design with Creative Exchange](#)

[Styling](#)

[Enable and configure the Asset Optimizer](#)

[Enable the Asset Optimizer globally](#)
[Change the optimization settings for a specific site](#)
[The Asset Optimizer](#)
[The grid settings](#)
[The grid systems](#)
[The grid definition](#)
[The grid classes](#)
[The HTML structure of pages and renderings](#)
[Page HTML](#)
[Rendering HTML](#)
[JavaScript](#)
[CSS](#)
[Headless](#)
[Introducing SXA data modeling](#)
[Enable SXA data modeling](#)
[JSON layout](#)
[JSON renderings](#)
[JSON variants](#)
[Add a JSON rendering and variant to your page](#)
[Switch to JSON](#)
[Add a JSON rendering](#)
[Add a JSON variant](#)
[Select a JSON variant](#)
[The JSON search parameters](#)
[Use the SXA Layout Service to model your pages in JSON](#)
[View your pages in JSON](#)
[Filter JSON data using search parameters](#)
[Trigger a JSON layout rendering](#)
[Best practices](#)
[Recommendations: Working with data sources and media](#)
[Clean up unused data sources](#)
[Do not put media items directly under the site's Media folder](#)
[Give site data sources meaningful names](#)
[Organize site data sources in folders](#)
[Run source field reports to help set the data source context](#)
[Recommendations: Extending SXA](#)
[Never put custom items in SXA-controlled branches of the tree](#)

[Do not modify OOTB SXA items](#)
[Do not replace the SXA MVC layout](#)
[Create an SXA module for your custom renderings](#)
[Follow Helix principles when adding functionality to your solution](#)
[Consider using existing renderings before building a new one](#)
[Consider cloning existing renderings before building a new one](#)
[Limit scope of fields linking to items](#)
[Recommendations: Sharing content](#)
[Consider using a shared site as the exclusive style container in a tenant](#)
[Consider defining your rendering variants in the shared site](#)
[Consider defining your page designs and partial designs in the shared site](#)
[Consider using delegated areas for pages that share content across sites](#)
[Consider creating a master site to clone for a rollout to new markets](#)
[Recommendations: Structuring pages](#)
[Design the page layout in partial designs](#)
[Put complex reusable structures in partial designs or composites](#)
[Remove unnecessary renderings from your site](#)
[Set placeholder restrictions](#)
[Do not use Standard Values to set presentation details for your pages](#)
[Use partial design inheritance](#)
[Prioritize changing the grid settings of a rendering over using splitters to change the layout of a rendering](#)
[Recommendations: Enhancing SXA performance](#)
[Set HTML caching settings for renderings](#)
[Limit the number of renderings on a page](#)
[Disable content testing for editors who do not require that functionality](#)
[Verify that you have enabled the Asset Optimizer](#)
[Recommendations: Working with themes](#)
[Do not modify platform themes](#)
[Assign custom styles to renderings](#)
[Clean up your styles folder](#)
[Organize custom style items in subfolders](#)
[Recommendations: Using renderings](#)
[Use the proper rendering for the job](#)
[Use the Snippet rendering for a group of renderings](#)
[Do not use the Plain HTML rendering for content that is edited by authors](#)
[Make sure the Rich Text field content is fully editable](#)

Sitecore Experience Accelerator (SXA) FAQ

Can you mix SXA and non-SXA websites within a single Sitecore instance?

Can you host a non-SXA site within an SXA tenant?

Can SXA pages contain non-SXA components?

Can I use SXA components outside of an SXA site?

Is there a limit on the number of sites that you can have in a single SXA installation?

Is the Sitecore PowerShell Extensions (SPE) required to be installed on every instance in the environment?

Where can I download the SPE WDP files required for Azure deployment?

What is Sitecore doing to prevent components from breaking between versions?