* AEM
* Apache Sling
* Java Content Repository (JCR)
* Open Service Gateway Initiative (OSGI)

Aem runs in 2 ways

**Author**

* Website Author logs in to siteadmin to create website
* Primary job to create pages from existing template
* Creating the content for the website using reusable component
* Components are reusable and pages are comprised multiple components
* Generally even ports are used for Author

Ex: - 4502

**Publish**

* The read only mode of website is called publish
* Generally odd ports are used for Publish

Ex: - 4503

**Classic UI and Touch UI**

<http://localhost:4502/welcome.html> used to access classic UI

To open an website with classic ui **[http://localhost:4502/cf#](http://localhost:4502/cf)** prefixis used

<http://localhost:4502/projects.html> used to access touch UI

To open an website with touch ui **<http://localhost:4502/editor.html>** prefixis used

Important point

The dialog creation for classic UI and Touch UI is different.

Accessing aem classic UI

<http://localhost:4502/libs/cq/core/content/welcome.html>

To directly go to page creation in site

<http://localhost:4502/siteadmin>

**AEM Folder structure**

1) The /apps folder is where you will store CQ5 elements such as Templates, Components, OSGi bundles, and static files. In order to start a new application/project, it is necessary to define a location for these elements. Typically, they are defined as a subfolder of the /apps folder (/apps/your project/components).

2) The /content folder contains the pages you create for the website using the templates. All the data in the components you add to the page are stored under the page's node in the content folder.

3) The /etc folder holds the designs, the common 'CSS' and the design level configurations.

4) The /libs folder includes Libraries and definitions that belong to the core of AEM. The sub-folders in /libs represent the out of the box AEM features as for example search or replication. The content in /libs should not be modified as it affects the way AEM works. Features specific to your website should be developed under /apps.

5) The /var folder includes files that change and are updated by the system; such as audit logs, statistics, event-handling. The sub-folder /var/classes contains the java servlets in source and compiled forms that have been generated from the components scripts.

6) The /tmp folder is the Temporary working area.

7) The /home folder includes users and group information and it also includes ACLs to determine what actions a user or group can take and where it can perform those actions.

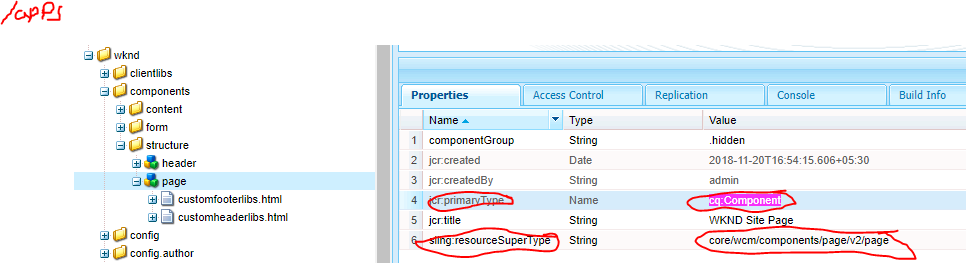
**Analysis on design**

**Page Component**

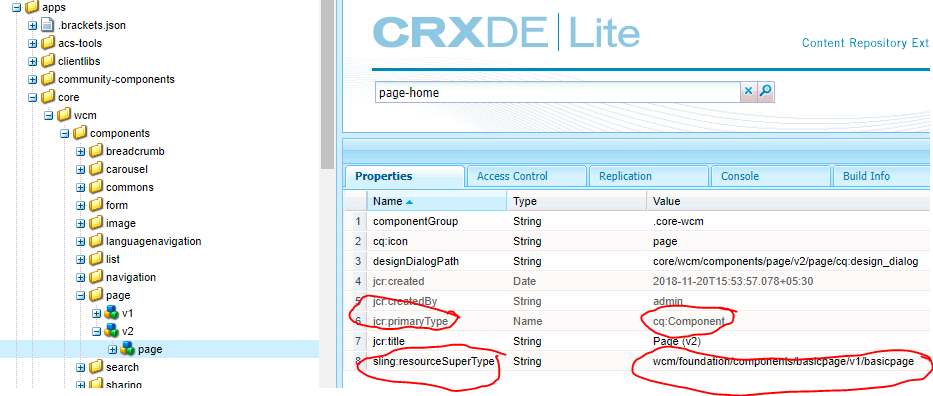
Page rendering component has property as **sling:resourceSuperType** which points to the base page.

That means page rendering component in extending properties from super pages which will be present either in the apps folder ultimately it will be extending from libs folder.

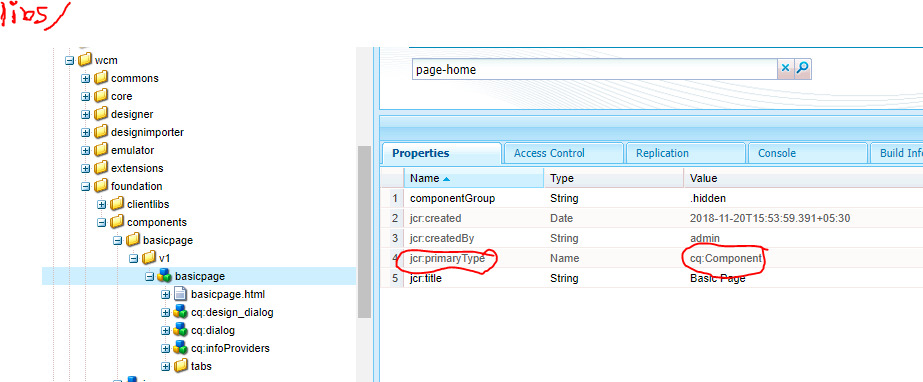
One thing to notice all these pages the **Jcr:primaryType** will be **cq:Component.**



The below page is super page for the above one. But the below page also extending from another page. As we can see **sling:resourceSuperType** property present.



The below one the final base page. Below one does not have **sling:resourceSuperType**



**Template**

Static templates are present in **/apps/{project folder}/templates**

A template is a node of type **cq:Template** and has the following properties and child nodes:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| . | cq:Template | Current template. A template is of node type cq:Template. |
| allowedChildren | String[] | Path of a template that is allowed to be a child of this template. |
| allowedParents | String[] | Path of a template that is allowed to be a parent of this template. |
| allowedPaths | String[] | Path of a page that is allowed to be based on this template.  Mean it defines the places in content (in page hirarchy) section where this template is allowed to use. |
| jcr:created | Date | Date of creation of the template. |
| jcr:description | String | Description of the template. |
| jcr:title | String | Title of the template. |
| ranking | Long | Rank of the template. Used to display the template in the User Interface. |
| jcr:content | cq:PageContent | Node containing the content of the template. |
| thumbnail.png | nt:file | Thumbnail of the template. |
| icon.png | nt:file | Icon of the template. |

**Note**

When we create a new page, the property “**cq:allowedTemplates**” with value from the template definition is created.

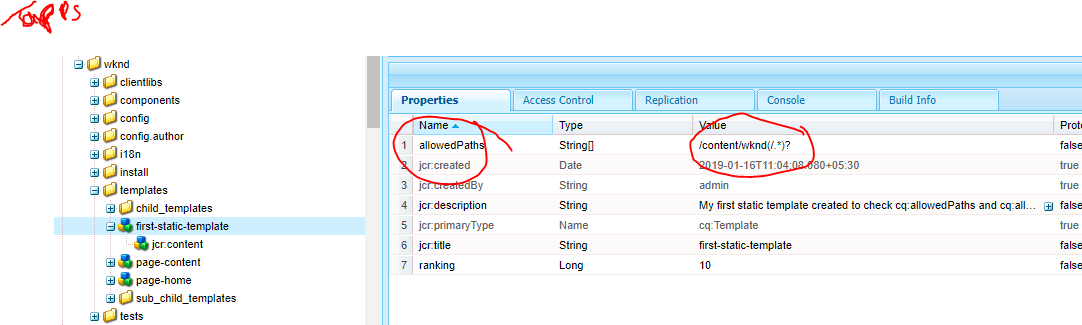
**cq:allowedTemplates** is a page property (in content/\*\*\*\*) location.

If the value of the property changes in template definitions, pages which are already created are not updated.

Once the page is created from static template you can't update structure/properties. You may need to write utility to update **cq:allowedTemplates** property in page.

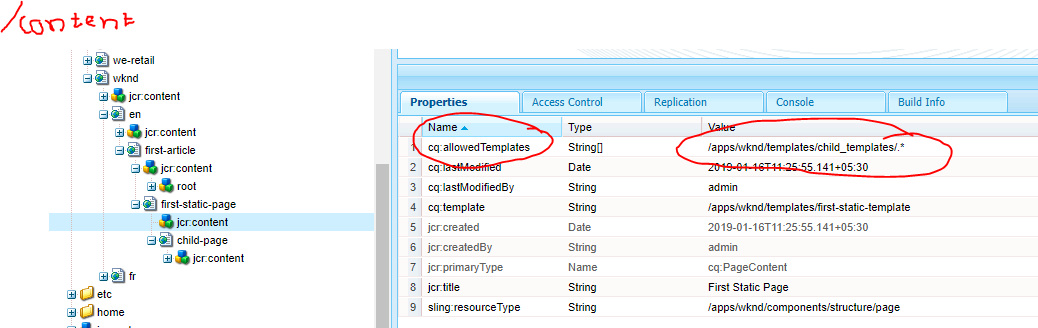
Difference between **allowedPaths, cq:allowedTemplates, allowedParents, allowedChildren**

**allowedPaths:**



1. **allowedPaths** is template level property.
2. The value of this property is of type array of string.
3. The value points to content (ex /content/wknd(/.\*)?) . Which means this template is available after wknd page onwards.
4. This property is available directly at template node level (not **jcr:content** level).

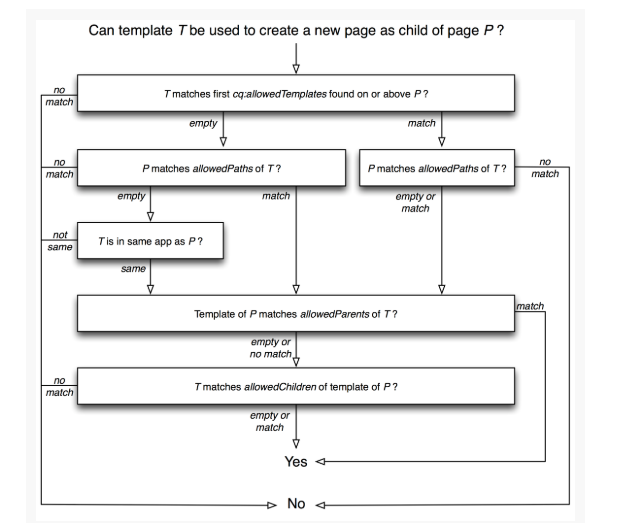
**cq:allowedTemplates:**



1. cq:allowedTemplates is a page level property
2. This property is available under **jcr:content** node.
3. The value of this property is of type array of string.
4. For dynamic templates the value will be pointing to **/conf/wknd/settings/wcm/templates**

If cq:allowedTemplates property is present it will verify the allowed templates are available in those locations which were configured against cq:allowedTemplates .Only the templates satisfies those locations will be made available to create pages.

**allowedParents** and **allowedChildren** property is at template level these two properties decide whether current template T can have which templates as parent and which templates as child.



Jcr:primaryTypes

|  |  |
| --- | --- |
| **Jcr:primaryType** **In** | **value** |
| Template | cq:Template |
| Jcr:content depends on what it holds | cq:PageContent,nt:unstructured |
| Pages | cq:Page |
| Components,page rendering components | cq:Component |
| Folders | sling:Folder,sling:OrderedFolder,nt:folder |
| Client library Folder | cq:ClientLibraryFolder |
| Initial ,policies,structure under dynamic template | cq:Page |

**OSGi (in simple words it a java jar file)**

Modular bundles, modular programming approach

Each module is called bundle

Bundle = jar + Manifest file (classes, interfaces, packages)

Bundles can started and stopped individually (this is not possible in typical web application)

Each of the provided bundles are responsible for each features present in aem (logging, replication)

Manifest file helps in managing the visibility between the bundles

Osgi implementation in aem is **Apache Felix**

On the top of apache Felix all the bundles are deployed

Other examples: - **Eclipse Equinox, Knopflerfish, ProSyst**

http://localhost:4502/system/console

<http://localhost:4502/system/console/bundles>

Bundles are stored in **crx-quickstart\launchpad**

**JCR (in simple words it’s a data store / database storing technique is different)**

Object database (supports both structured and unstructured data)

Data stored in the form of nodes and properties

Every property has a key , value and type. Type represents what data type.

Adobe implementation of JCR is CRX (content repository extreme) but in aem 6.4 the implementation is **Apache Jackrabbit Oak**

File system is best for storing unstructured data like images, documents, videos etc

Database is best for storing structured data in the form of relation tables

But JCR is best in both.

<http://localhost:4502/crx/de/index.jsp>

http://localhost:4502/crx/explorer

JCR uses hierarchical model for storing data

Everything starts with root (/)

Child

Child

Child

Creation of new page also creates nodes and properties

JCR is a specification (just like an interface in java)

Other examples of JCR implementation is Apache Jackrabbit

Data is stored in physical **crx-quick-start** folder as **TAR** files this not in human readable format

**JCR** (JSR-170) is the a standard Java **API** for content repositories. It provides an interface for Java applications to interact with content repositories, however it does not provide a repository implementation.

Apache [**Jackrabbit**](http://jackrabbit.apache.org/) is an **open source** content repository [**implementation**](http://jackrabbit.apache.org/jcr-api.html) that fully implements the JCR API.

Adobe **CRX** is the commercial content repository component used in the AEM, which uses some elements of Jackrabbit (e.g. some of the [security APIs](http://jackrabbit.apache.org/api/2.4/org/apache/jackrabbit/api/security/user/package-summary.html)). CRX provides additional features such as [development tools](http://dev.day.com/docs/en/crx/current/developing/development_tools/developing_with_crxde.html) & [clustering capabilities](http://dev.day.com/docs/en/crx/current/administering/cluster.html) and has its own [storage mechanism](http://dev.day.com/content/ddc/blog/2008/11/tarpm.html) which differs from the Jackrabbit implementation.

Benefits of crx

Browsing capability to view nodes and properties.

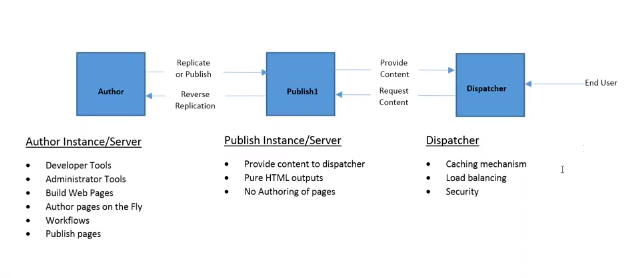
Apache Sling

The Resource is one of the central parts of Sling. Extending from JCR's Everything is Content, Sling assumes Everthing is a Resource. Thus Sling is maintaining a virtual tree of resources, which is a merger of the actual contents in the JCR Repository and resources provided by so called resource providers. By doing this Sling fits very well in the paradigm of the REST architecture.

Sling is used to access JCR over web

For Replication and reverse replication

<http://localhost:4502/miscadmin>



Replicate entire tree is done through **Tree Activation** option in replication node (need to double click) this is under **miscadmin** section

**Options available in Dispacher.any file**

**/farms 🡪 this section is added for each organization example Michelin this is the parent section under one firm we can have multiple website.**

**/website 🡪**

**Under farm we can multiple website. Example: Bfgoodrich and Uniroyal**

**/clientheaders 🡪** **Request headers that should be forwarded to the remote server**

**/virtualhosts 🡪 Domain address for the site. “\*” means localhost**

**/renders 🡪 this will contain all the publishers details each with its own unique section .each of those sections will contain domain and port details. Load balance among multiple publishers happens here.**

**Example: /rend01**

**/renders**

**{**

**/rend01**

**{**

**# Hostname or IP of the render**

**/hostname "127.0.0.1"**

**# Port of the render**

**/port "4503"**

**# Connect timeout in milliseconds, 0 to wait indefinitely**

**# /timeout "0"**

**}**

**}**

**/filter 🡪 this section is used to allow or deny specific content which are being requested to the publish environment**

**Filter will have unique sections with rules.**

**Ex:**

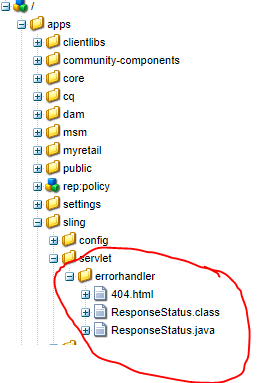
**/0011 { /type "allow" /url "/admin/\*" } # allow servlet engine admin**

**/0012 { /type "allow" /url "/crx/\*" } # allow content repository**

**/0013 { /type "allow" /url "/system/\*" } # allow OSGi console**

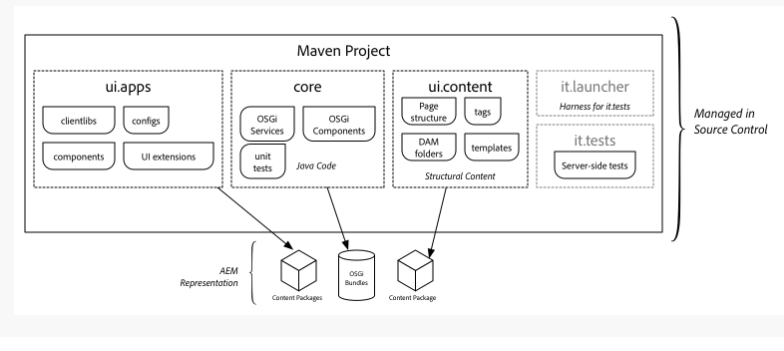
**/cache 🡪 this section decides what needs to be cached and where it should be cached.**

**Default Error Handler in AEM**



**Command to install project to AEM**

**mvn -PautoInstallPackage -Padobe-public clean install**



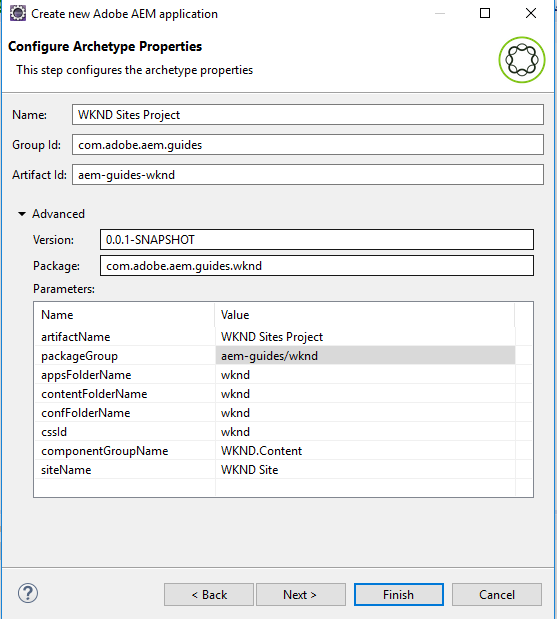
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate \

 -DarchetypeGroupId=com.adobe.granite.archetypes \

 -DarchetypeArtifactId=aem-project-archetype \

 -DarchetypeVersion=15 \

 -DarchetypeCatalog=https://repo.adobe.com/nexus/content/groups/public/

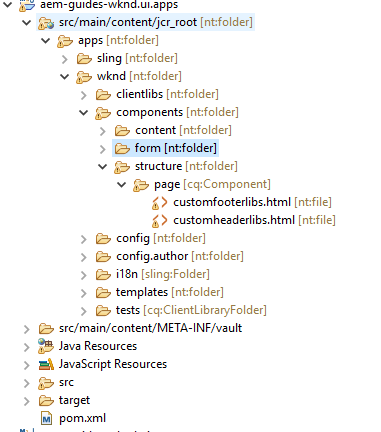


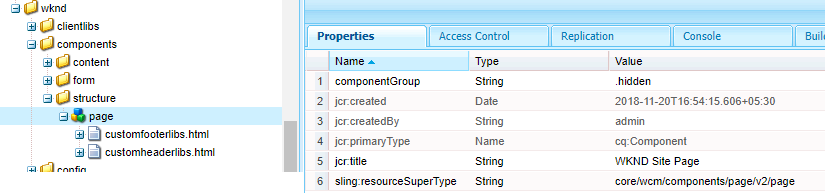
**componentGroupName property**

**/apps/wknd/components/content/breadcrumb (check for componentGroup property)**

**Structure folder under components**

 Any component added into the structure folder indicates that the component is meant to be used when constructing a template, and not to be used when authoring a page.



****

|  |  |
| --- | --- |
|  |  |

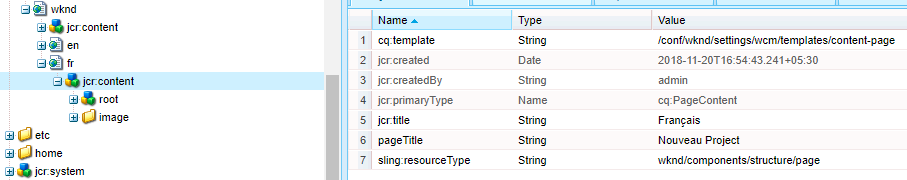
**What are the differences between sling:resourceType and sling:resourceSuperType ?**

**sling:resourceSuperType:**

**It is used to achieve inheritance in cq. When set, it inherits the specified component to this component. This is also called component proxied.**

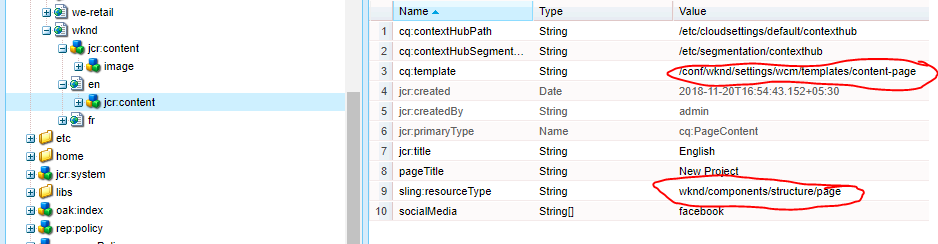
**sling:resourceType:**

**It is a path, which locates the script to be used for rendering the content. Path used can be absolute or relative. This happens in content folder for pages. Pages points to page rendering component.**

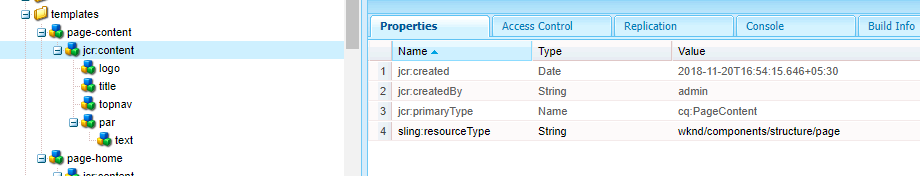


**At content level (where all the pages are stored) both template and page rendering component (it will be present in structure folder of component) information is stored**

**So page gets the rendering component information from template**



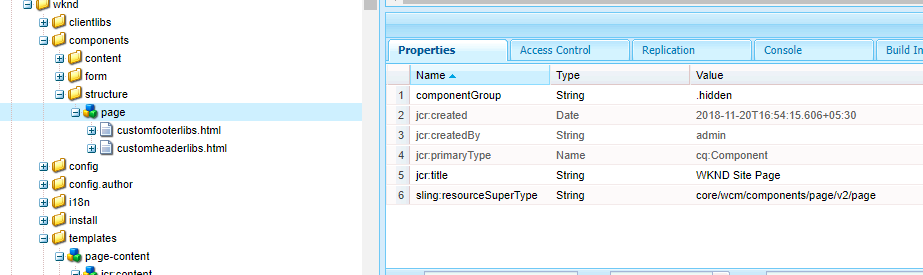
Templates points to the page rendering component sling: resourceType tells what to render.



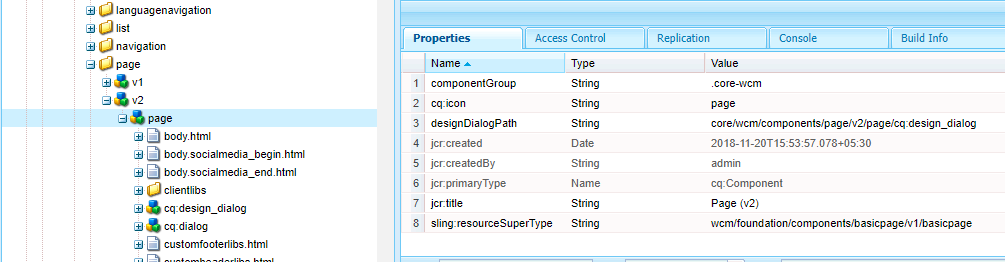
The page rendering component extends (inherits) the specified component to this component.

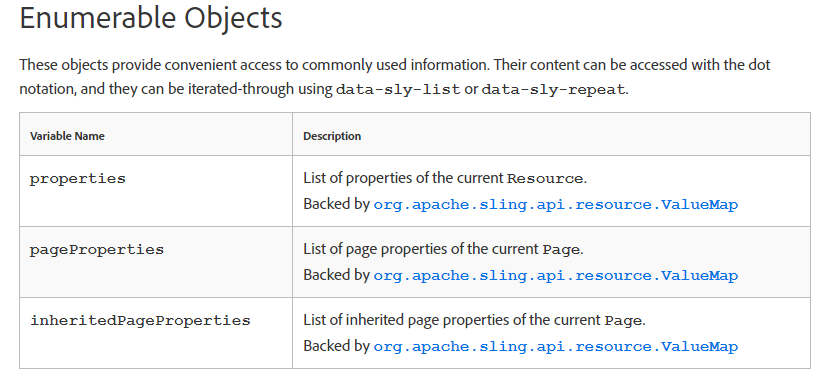
That’s why sling: resourceSuperType property key used.

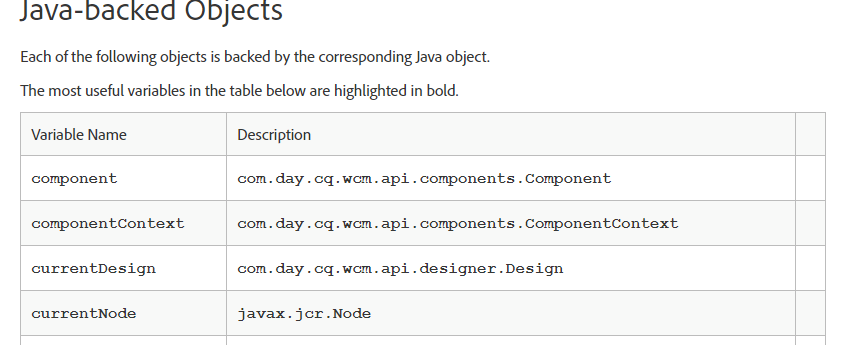
This points to /apps/core/wcm/components/page/v2/page



The page in /apps/core/wcm/components/page/v2/page gets all the resources from /libs/wcm/foundation/components/basicpage/v1/basicpage











Template folders

There are two template folders

1. /apps/wknd/templates (static templates )
2. /conf/wknd/settings/wcm/templates(editable templates)

Editable and Static Templates

AEM now offers two basic types of templates:

**Editable Templates**

Can be created and edited by template authors using the Template console and editor. The Template console is accessible in the General section of the Tools console.

After the new page is created a dynamic connection is maintained between the page and the template. This means that changes to the template structure and/or locked content will be reflected on any pages created with that template. Changes to the unlocked (i.e. initial) content will not be reflected.

Use content policies, which you can define these from the template editor, to persist the design properties. Design mode within the page editor is no longer used for editable templates.

Are stored under /conf.

**Static Templates**

Static templates have been available for several versions of AEM.

They are provided by your developers, so they cannot be created or edited by authors.

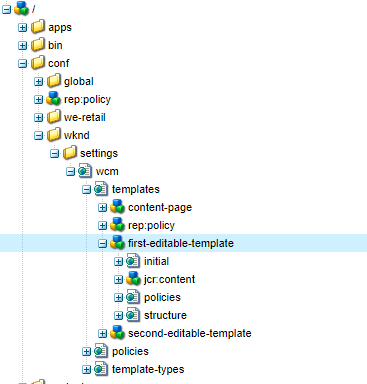
Are copied to create the new page, but no dynamic connection exists after this (though the template name is registered for information).

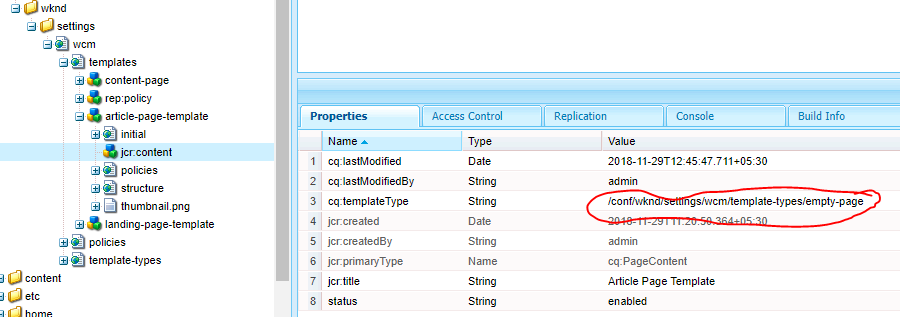
Use Design Mode to persist design properties.

Because editing static templates is the exclusive task of a developer, please see the developer document Page Templates - Static for more information.

Link to go to editable template console

<http://localhost:4502/libs/wcm/core/content/sites/templates.html/conf>





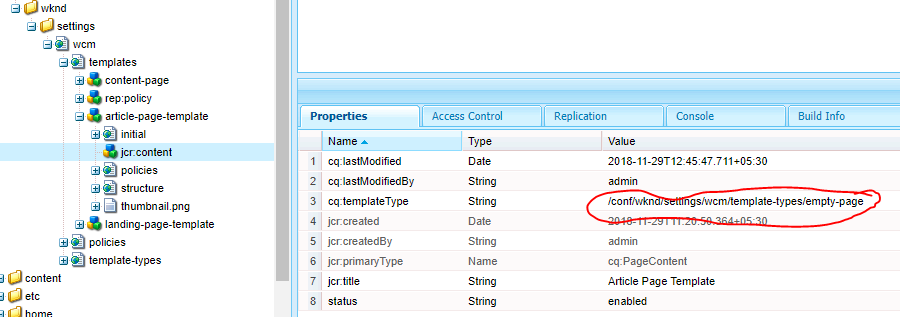
All editable templates will contain 3 things

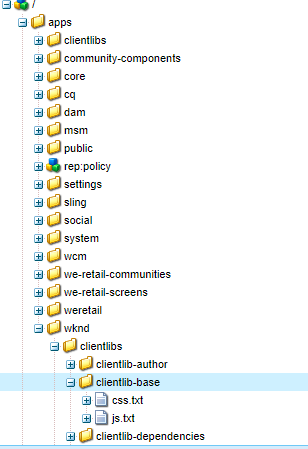
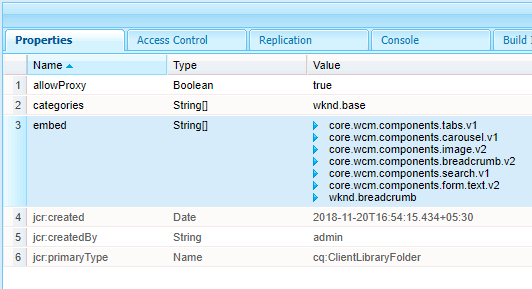
1. **Structure** - defines components that are a part of the template. These will not be editable by content authors.
2. **Initial Content** - defines components that the template will start with, these can be edited and/or deleted by content authors
3. **Policies** - defines configurations on how components will behave and what options authors will have available.

Every editable template uses a property to refer the template type (template type is also a kind of template)

Template types also has 3 sections similar to the template.

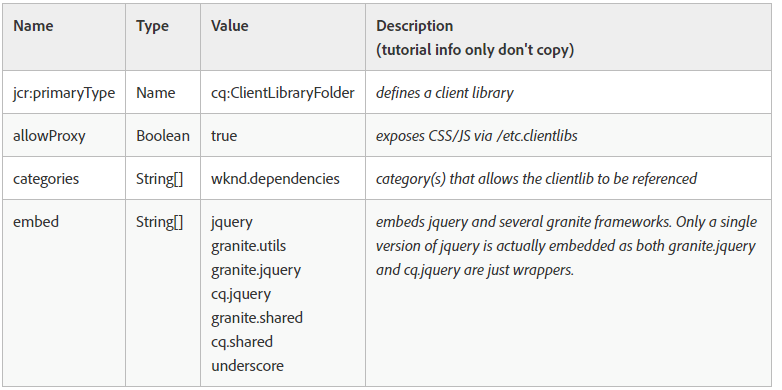
**Client libs**



/apps/wknd/clientlibs/clientlib-base

Each of the client libs embedded in clientlib-base project are from /apps/core/wcm/components (from core components)



Jcr:primaryType = cq:ClientLibraryFolder

This property represents it as client library

allowProxy = true means to access css and js prefix will be /etc.clientlibs

embed = this property is used to include other client libraries instead js and css.

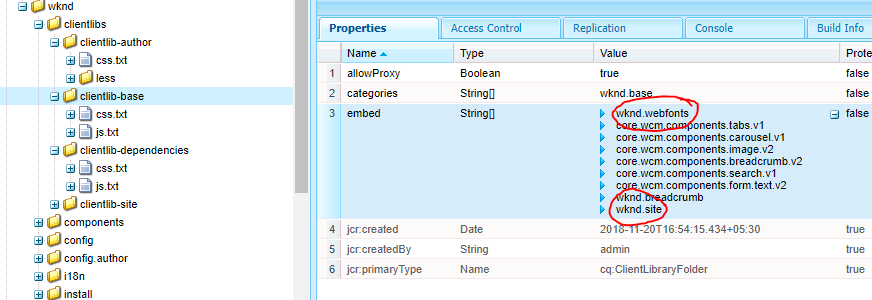
Categories= this property only used while referencing the current client lib.

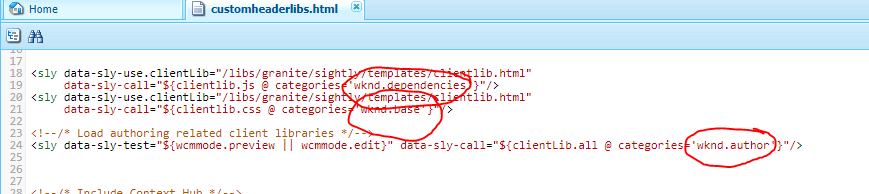
<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html"

data-sly-call="${clientlib.css @ categories='wknd.base'}"/>

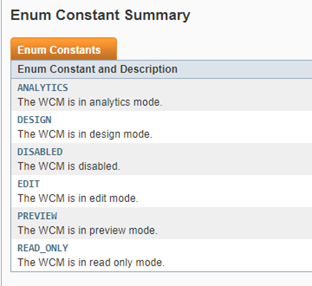
The above line is used to include clientlibs in to page rendering component. Clientlib is just an alias name for accessing the css.

Few observations





wknd.site is not added to customheaderlibs.html as it’s already included in wknd.base as embedded property



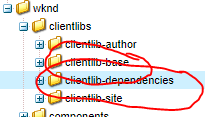
Generally css are loaded in the header of page component and js are loaded in the footer of the page component.

|  |
| --- |
|  |
|  | <script type="text/javascript" src="[/etc.clientlibs/wknd/clientlibs/clientlib-dependencies.js](http://localhost:4502/etc.clientlibs/wknd/clientlibs/clientlib-dependencies.js)"></script> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | <link rel="stylesheet" href="[/etc.clientlibs/wknd/clientlibs/clientlib-base.css](http://localhost:4502/etc.clientlibs/wknd/clientlibs/clientlib-base.css)" type="text/css">  <script type="text/javascript" src="/etc.clientlibs/wknd/clientlibs/clientlib-base.js"></script> |

**Note**

The name of the js and css file is decided by the client-lib node name.

The main takeaway is that the CSS and Javascript is dynamically loaded from a path that starts with **/etc.clientlibs**. The client libraries are stored beneath **/apps/wknd** to make it easier from an organizational standpoint as it is in the same directory as our component code. It is critical on the publish side that the client libraries are **not** served from /apps as this path should be restricted for security reasons using the [Dispatcher filter section](https://docs.adobe.com/docs/en/dispatcher/disp-config.html#Example%20/filter%20section). The [allowProxy property](https://docs.adobe.com/docs/en/aem/6-3/develop/the-basics/clientlibs.html#Locating%20a%20Client%20Library%20Folder%20and%20Using%20the%20Proxy%20Client%20Libraries%20Servlet) of the client library ensures the CSS and JS are served from /etc.clientlibs.



<http://localhost:4502/libs/granite/ui/content/dumplibs.html>

http://localhost:4502/libs/granite/ui/content/dumplibs.test.html

<http://localhost:4502/libs/granite/ui/content/dumplibs.validate.html>

<http://localhost:4502/libs/granite/ui/content/dumplibs.rebuild.html>

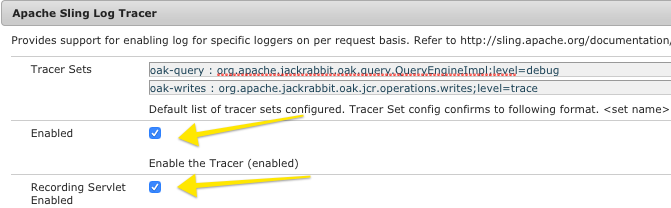
Aemfed

A key part of aemfed is the ability to relay errors directly to the command line. To enable this behavior we need to update the configuration for Sling Log Tracer.

<http://localhost:4502/system/console/configMgr>

Config manager is used for configuring all the default values

Navigate to <http://localhost:4502/system/console/configMgr> and search for **Apache Sling Log Tracer**. Update the configuration to **Enabled** and **Recording Servlet Enabled**: 



Caution:

Apache Sling Log Tracer version 1.0.0 or newer should be used and should only be enabled in development and other non-production environments.

To install aemfed:

npm install aemfed --global

**Run the following command to start aemfed against an aem instance running on localhost:4502:**

We have run this command in project root directory

aemfed -t "http://admin:admin@localhost:4502" -e "\*\*/\*\_\_\_jb\_+(old|tmp)\_\_\_" -w "ui.apps/src/main/content/jcr\_root/"

After running above command

---------------------------------------

[Browsersync] Proxying: http://localhost:4502

[Browsersync] Access URLs:

 --------------------------------------

       Local: http://localhost:3000

    External: http://192.168.1.152:3000

 --------------------------------------

          UI: http://localhost:3001

 UI External: <http://localhost:3001>

After starting aemfed we can access the localhost:3000

<http://localhost:3000/content/wknd/en/first-article.html?wcmmode=disabled>

**Breadcrumb**

Configurable start level

Option to show hidden navigation items

Exclude the current page from the breadcrumb

**Content Fragment**

Allow for article text (copy) to be created and managed independently of a page

Promotes reuse and variations for cross-channel

**Image**

Smart loading of optimal rendition

In-place editing, cropping, rotating, and resizing

Image title, description, accessibility text and link

List

**Multiple sources:**

List page children

List tagged items

List query result

List static items

Ordering, pagination and limit

Styles

**Text**

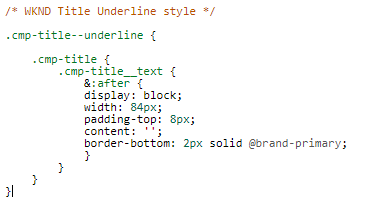
In-place editing

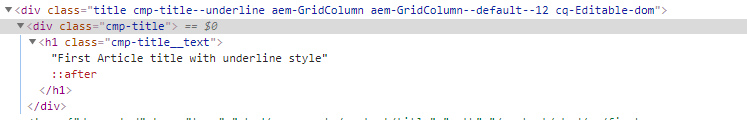
Rich Text authoring

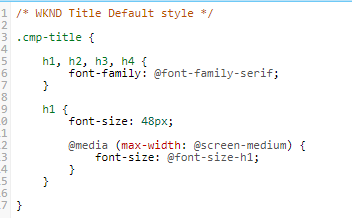
**Title**

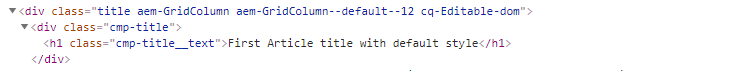
In-place editing

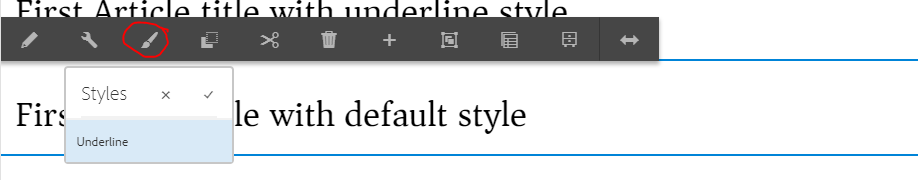
Use the Page title with option to override the text





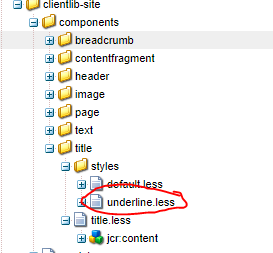


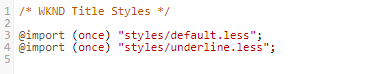




Applying style in page level. In the drop down all configured styles will show up.

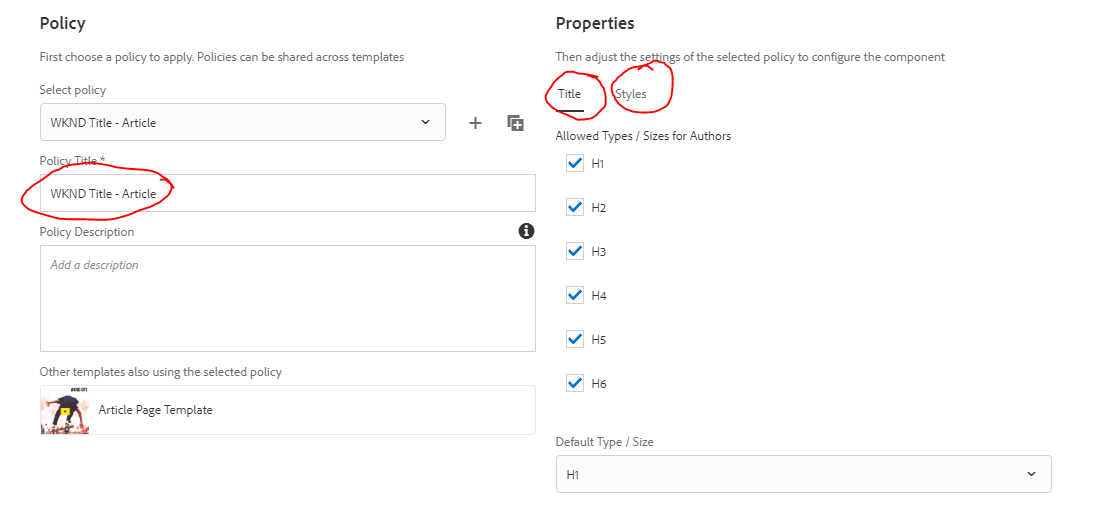
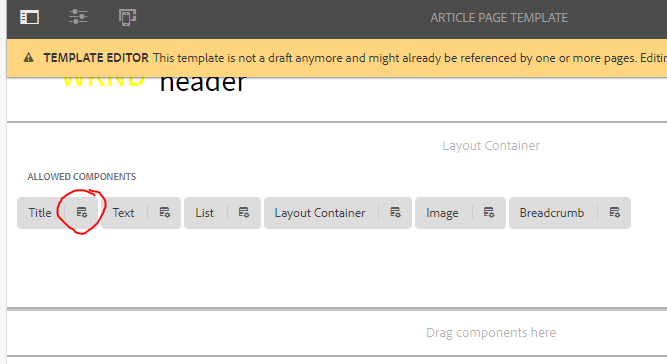
**How to add a new style option in a component**

 Add new css file for new style option. The top level css class is applied to the outer div.

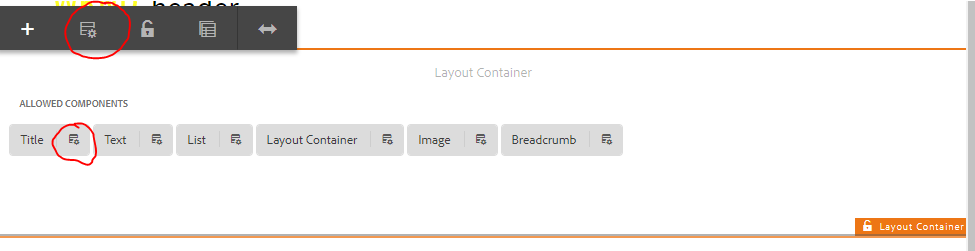


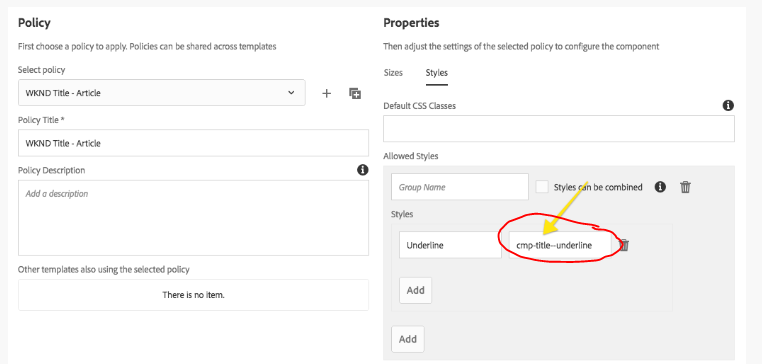
Import the style in **title.less** file

Go to the editable template and below icon to add new policy at component level.



Give a policy name and then select all allowed types sizes.

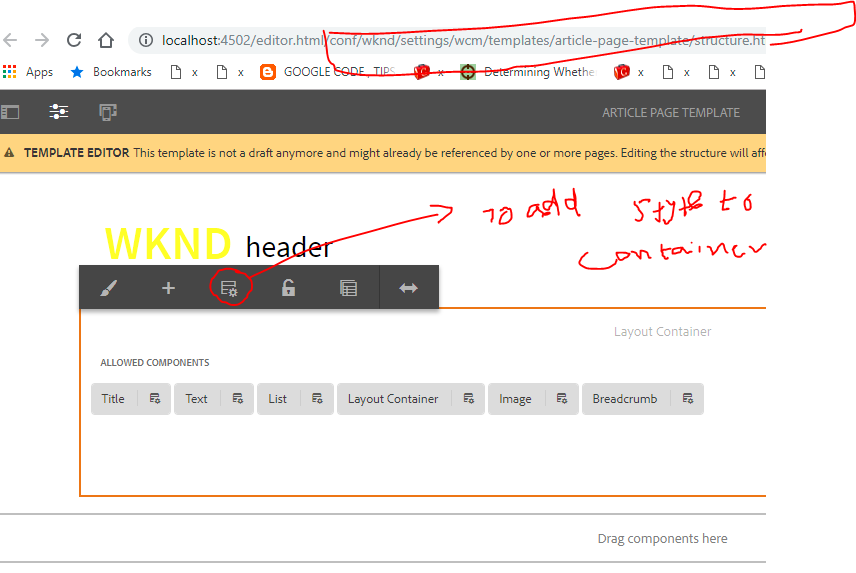




We have to provide the top level css class name present in the new css file. Give a name to select in the dropdown while applying css in authoring mode.

While applying css in layout container level we have to select the profile and add the style class.

By changing the layout container will not the impact the changes on the already created pages.



**Content Fragment Component**

You will notice that each paragraph of the Content Fragment has a drop-zone to embed additional components within the Content Fragment. You will also notice that the content of the Content Fragment is read-only. This is by design so that the Content Fragment can only be updating by editing the original fragment.

Stored as asset.

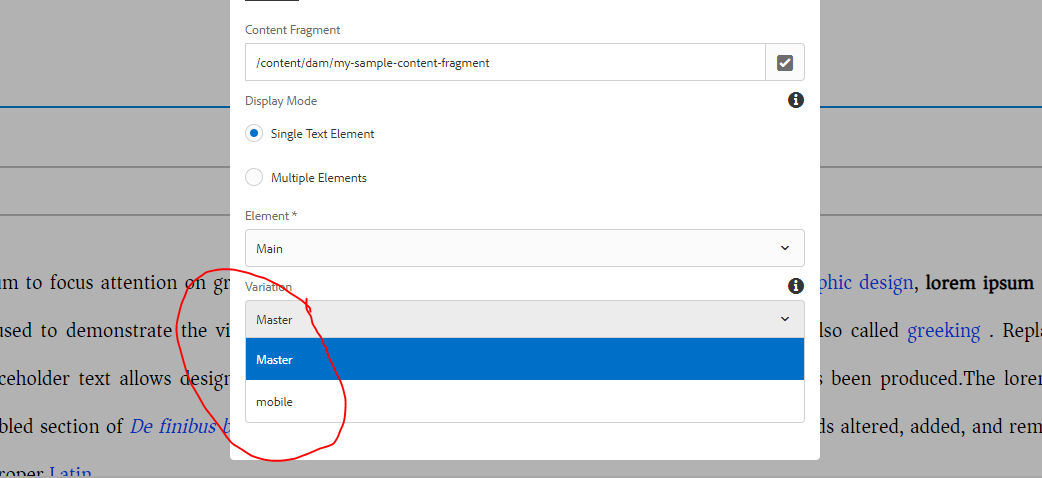
Content asset can contain text elements and might include references to other assets.

Content authors can create content fragment before it is being authored in a page.

We can create multiple variations of content fragment and we can change them

We can have hierarchy of content fragment. We can put content fragment within content fragment.



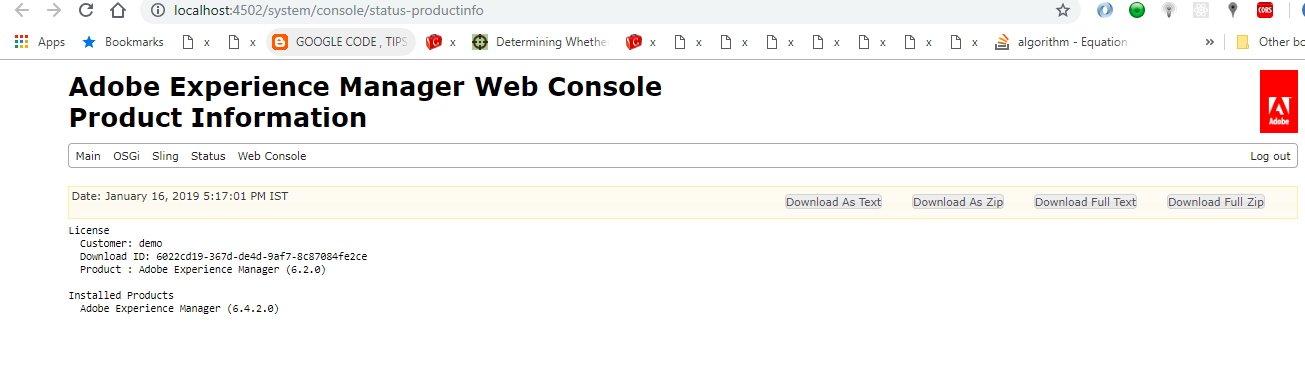


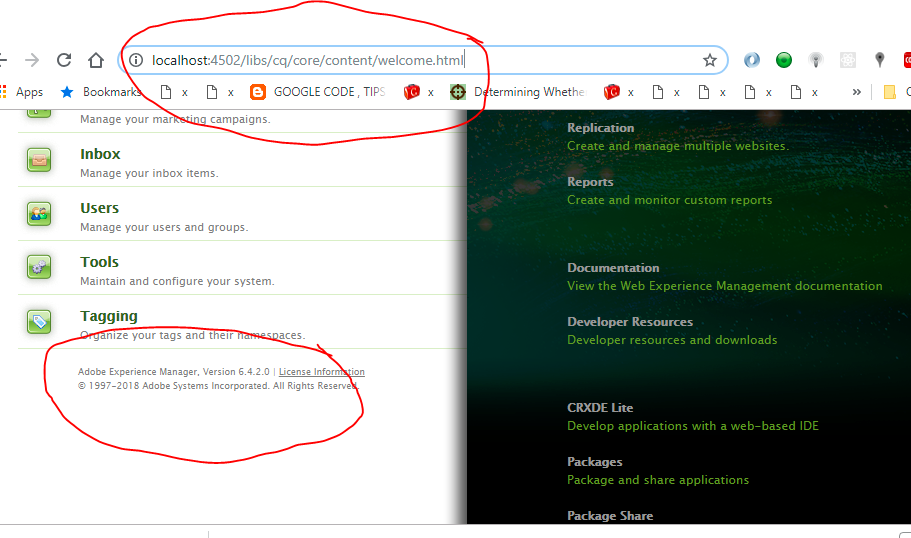
**To see version of aem**

<http://localhost:4502/system/console/status-productinfo>

<http://localhost:4502/libs/cq/core/content/welcome.html>

same as <http://localhost:4502/welcome.html>





### URL Decomposition

In Sling, processing is driven by the URL of the user request. This defines the content to be displayed by the appropriate scripts. To do this, information is extracted from the URL.

If we analyze the following URL:

|  |  |
| --- | --- |
| **1** | [**http://myhost/tools/spy.printable.a4.html/a/b?x=12**](http://myhost/tools/spy.printable.a4.html/a/b?x=12) |

We can break it down into its composite parts:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| protocol | host | content path | selector(s) | extension |  | suffix |  | param(s) |
| http:// | myhost | tools/spy | .printable.a4. | html | / | a/b | ? | x=12 |

**Slightly**

**Sightly comment**:

<!--/\* comment \*/-->

Benefits of using slightly comment is that in page source it will not be visible.

**Expression**

${} at run time expression is evaluated

Can be used inside attributes, element content, comments.

Example ${properties.jcr:title}

${! currentPage.hasChild}

${varOne || vartwo}

**Sly Element**

It removes itself from the final output.

<div data-sly-include=”header.html”> </div>

o/p🡪<div> header.html file output </div>

<sly data-sly-include=”header.html”> </sly>

o/p🡪 header.html file output

**Sightly Attributes**

**data-sly-text** 🡪 It replaces the tag content with the value of the expression.

Ex: <div data-sly-text =”${currentPage.title}”>**Page Title**</div>

**Page Title** will be replaced with currentPage.title value.

**data-sly-list** 🡪 repeats the children element. The attribute **data-sly-list** is applied at parent tag.

ex: <ul data-sly-list="${currentPage.listChildren}">

<li> index: ${itemList.index} value: ${item.title} </li>

</ul>

**item**: The current item in the iteration.

**itemList**: Object holding the following properties:

**index**: zero-based counter (**0..length-1**).

**count**: one-based counter (**1..length**).

**first**: **true** if the current item is the first item.

**middle**: **true** if the current item is neither the first nor the last item.

**last**: **true** if the current item is the last item.

**odd**: **true** if **index** is odd.

**even**: **true** if **index** is even.

Defining an identifier on the data-sly-list statement allows you to rename the itemList and item variables. item will become <variable> and itemList will become <variable>List.

<dl data-sly-list.child="${currentPage.listChildren}">

<li> index: ${childList.index} value: ${child.title} </li>

</dl>

**data-sly-repeat:**

With data-sly-repeat you can repeat an element multiple times based on the list that is specified. In this case the tag where this attribute is applied will repeat itself.

<ul >

<li data-sly-repeat="${currentPage.listChildren}" > index: ${itemList.index} value: ${item.title} </li>

</ul>

**data-sly-test:**

Conditionally shows or removes the element (tag) and its content if the condition is true.

For example, the p element and its content will only be rendered if isShown is true:

<p data-sly-test="${isShown}">text</p>

**data-sly-include:**

This attribute is used to include other html/jsp file content.

<sly data-sly-include=”myfile.html”> </sly>

**data-sly-resource:**

This is used to include component .Includes the result of rendering the indicated resource through the sling resolution and rendering process.

**<article data-sly-resource="path/to/resource"></article>**

**<article data-sly-resource="${ @ path='path/to/resource'}"></article>**

**<article data-sly-resource="${'resource' @ prependPath='my/path'}"></article>**

**<article data-sly-resource="${'my/path' @ appendPath='resource'}"></article>**

Add (or replace) a selector:

<article data-sly-resource="${'path/to/resource' @ selectors='selector'}"></article>

Add, replace or remove multiple selectors:

<article data-sly-resource="${'path/to/resource' @ selectors=['s1', 's2']}"></article>

Add a selector to the existing ones:

<article data-sly-resource="${'path/to/resource' @ addSelectors='selector'}"></article>

Remove some selectors from the existing ones:

<article data-sly-resource="${'path/to/resource' @ removeSelectors='selector1'}"></article>

Remove all selectors:

<article data-sly-resource="${'path/to/resource' @ removeSelectors}"></article>

Changes the WCM mode:

<article data-sly-resource="${'path/to/resource' @ wcmmode='disabled'}"></article>

path/to/resource = path to resource can be a relative path or full path. If in cotes then relative or else full path.

wcmmode='disabled' will disable edit mode of the component.

**data-sly-use**:

Initializes a helper object (defined in JavaScript or Java) and exposes it through a variable.

Initialize a JavaScript object, where the source file is located in the same directory as the template. Note that the filename must be used:

|  |
| --- |
| **<div data-sly-use.nav="navigation.js">${nav.foo}</div>** |

Initialize a Java class, where the source file is located in the same directory as the template. Note that the classname must be used, not the file name:

|  |
| --- |
| **<div data-sly-use.nav="Navigation">${nav.foo}</div>** |

Initialize a Java class, where that class is installed as part of an OSGi bundle. Note that its fully-qualified class name must be used:

|  |
| --- |
| **<div data-sly-use.nav="org.example.Navigation">${nav.foo}</div>** |

**data-sly-template and data-sly-call:**

**data-sly-template** defines a template and with the help of

**data-sly-call** is used to call a template**.**

Define a static template and then call it:

|  |
| --- |
| **<template data-sly-template.one>blah</template>**  **<div data-sly-call="${one}"></div>** |

o/p will be : <div> blah </div>

Define a dynamic template and then call it with parameters:

|  |
| --- |
| **<template data-sly-template.two="${ @ title}"><h1>${title}</h1></template>**  **<div data-sly-call="${two @ title=properties.jcr:title}"></div>** |

**o/p: <div>**

**<h1> Title </h1>**

**<div>**

**Common usage:**

Templates located in a different file, can be initialised with **data-sly-use**. Note that in this case **data-sly-use** and **data-sly-call** could also be placed on the same element:

|  |
| --- |
| **<div data-sly-use.lib="templateLib.html">**  **<div data-sly-call="${lib.one}"></div>**  **<div data-sly-call="${lib.two @ title=properties.jcr:title}"></div>**  **</div>** |

**example**

data-sly-use call the template in different file and assigns to clientLib variable. clientlib.js is used to call the js template with argument categories variable. same for css one.

<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html"

data-sly-call="${clientlib.js @ categories='wknd.dependencies'}"/>

<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html"

data-sly-call="${clientlib.css @ categories='wknd.base'}"/>

Template recursion is supported:

|  |
| --- |
| **<template data-sly-template.nav="${ @ page}">**  **<ul data-sly-list="${page.listChildren}">**  **<li>**  **<div class="title">${item.title}</div>**  **<div data-sly-call="${nav @ page=item}" data-sly-unwrap></div>**  **</li>**  **</ul>**  **</template>**  **<div data-sly-call="${nav @ page=currentPage}" data-sly-unwrap></div>** |

**data-sly-unwrap:**

Removes the host element from the generated markup while retaining its content.

|  |
| --- |
| **<p data-sly-use.nav="navigation.js">Hello World</p>** |

will produce

|  |
| --- |
| **<p>Hello World</p>** |

Whereas this,

|  |
| --- |
| **<p data-sly-use.nav="navigation.js" data-sly-unwrap>Hello World</p>** |

will produce

|  |
| --- |
| **Hello World** |

It is also possible to conditionally unwrap an element:

|  |
| --- |
| **<div class="popup" data-sly-unwrap="${isPopup}">content</div>** |

**data-sly-attribute:**

**data-sly-attribute**: Adds attributes to the host element. Only benefit is we can still keep the static text in second approach.

|  |
| --- |
| **<div title="${properties.jcr:title}"></div>** |

is equivalent to

|  |
| --- |
| **<div title="Lorem Ipsum" data-sly-attribute.title="${properties.jcr:title}"></div>** |

**data-sly-element:**

**data-sly-element**: Replaces the element name of the host element.

For example,

|  |
| --- |
| **<h1 data-sly-element="${titleLevel}">text</h1>** |

Replaces the **h1** with the value of **titleLevel**.

For security reasons, **data-sly-element** accepts only the following element names:

|  |
| --- |
| **a abbr address article aside b bdi bdo blockquote br caption cite code col colgroup**  **data dd del dfn div dl dt em figcaption figure footer h1 h2 h3 h4 h5 h6 header i ins**  **kbd li main mark nav ol p pre q rp rt ruby s samp section small span strong sub**  **sup table tbody td tfoot th thead time tr u var wbr** |

To set other elements, XSS security must be turned off (**@context='unsafe'**).

**Sling Post servlet**

Modify content in a JCR repository underlying Sling, you have multiple options, two of which are **WebDAV** and the Sling default POST Servlet also called the **SlingPostServlet.** We can create, modify, copy, move, delete, import - content through the SlingPostServlet.

**What is content?**

Content means some data to be stored in the JCR repository to be later used as the basis for some presentation. In this sense Content is a rather conceptual term. Item is the name of the parent interface of the JCR Node and Property interfaces. When speaking of *Items we mean some actual data stored in the repository ignoring whether the data is actually stored as a Node with child nodes and properties or just a single Property.*

**Item** is parent interface of **Note** and **Property** interface.

## **[multipart/form-data POSTs](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "preface-multipart-form-data-posts)**

Sometimes you might want to have the content modifications applied in a certain order. This is particularly interesting if you use fields to create child nodes and if you want to stipulate a certain child node order based on the form fields.

In this case, ensure you are submitting the POST request using multipart/form-data encoding. This preserves the order of parameter application according to the original HTML form. To this avail, ensure to always include the enctype="multipart/form-data" attribute with the <form> tag.

This support requires Sling Engine 2.1.0 and the Sling Default Post Servlet 2.0.6.

Note:

enctype="multipart/form-data” should be used to create nodes in the order with which it is mentioned in form.

**SlingPostServlet Operations**

The SlingPostServlet is actually just a front-end to the actual operations. To select the actual operation to execute, the ***:operation*** request parameter is used. Out of the box, the SlingPostServlet supports the following operations:

property not set or empty -- Create new content or modify existing content

delete -- Remove existing content

move -- Move existing content to a new location

copy -- Copy existing content to a new location

import -- Import content structures from JSON/XML/Zip

nop -- Explicitly requests to do nothing and just sets the response status

checkin - Check in a versionable node

checkout - Check out a versionable node

**Note:**

All these operations always operate on the resource of the request as returned by SlingHttpServletRequest.getResource(). Some operations require additional parameters to be set to operate completely.

Please note that operations are mutually exclusive. For a single POST request only one operation may be executed. Operations also only consume the request parameters as described below. Any excess parameters are silently ignored.

`:applyTo` value is '\*' then the operation applies to all the children of the resolved parent resource. This can be used to act on all the children of a resource without having to specify the path of each individual child resource.

### [Content Creation or Modification](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "content-creation-or-modification)

The simplest and most common use case, probably, is content creation and modification. We already saw an example above in the quickstart section. In this section we elaborate more on the concrete stuff.

First, the request URL indicates the actual repository node to be handled. If the URL addresses an existing node, the request parameters just provide values for the properties to be set on the existing node.

If the resource of the request is a synthetic resource, e.g. NonExistingResource or StarResource, a new item is created. The path (including name) of the item to be created is derived from the resource path:

* If the resource path ends with a /\* or / the name of the item is automatically created using a name creation algorithm taking into account various request parameters.
* Otherwise the resource path is used as the path and name of the new item.

In both cases the path may still include selectors and extensions, which are cut off the path before finding out, what to do.

| **Resource Path** | **Item path** |
| --- | --- |
| /content/new | /content/new |
| /content/new.html | /content/new |
| /content/new.print.a4.html | /content/new |
| /content/ | /content/xxx where xxx is a generated name |
| /content/\* | /content/xxx where xxx is a generated name |
| /content/\*.html | /content/xxx where xxx is a generated name |
| /content/\*.print.a4.html | /content/xxx where xxx is a generated name |

**Note:**

In the mentioned path node already exist only the properties are updated.

In case the node is not there (i.e. the path ends / or /\*) then the name of the node is dynamically generated.

If with same field name multiple entries are submitted in an html form with different values then multi-field property is created.

In case properties provided are there, they will be modified or else new properties will be created.

Below properties are automatically set

* created and jcr:created are set to the node creation time, as a Date value.
* lastModified, jcr:lastModified are set to the node modification time, as a Date value.
* createdBy and jcr:createdBy are set to the name of the user who created the node.
* lastModifiedBy, jcr:lastModifiedBy are set to the name of the user who modified the node.

**File Uploads**

File uploads are typically done using the <input type="file""/> element of an HTML form and ensuring the correct form encoding. The SlingPostServlet handles uploaded files specially, in that the file data is not simply written into a property, but a node is actually created with three properties:

* jcr:data -- The actual file contents
* jcr:lastModified -- The time stamp of processing the uploaded file
* jcr:mimeType -- The MIME type from the original file submission (if contained in the file body part) or derived from the original file name

The name of the node is either taken from the parameter name or if the name is \* from the name of the uploaded file.

The primary node type of the uploaded file is selected using the following algorithm:

* If a `@TypeHint suffixed parameter (see below for a description) is present check whether the value is a known non-mixin node type. If so, the node is created with this primary node type.
* If a @TypeHint suffixed parameter is not present or the value does not denote an existing non-mixin node type, the node will be created as an nt:file node if the parent node is of type nt:folder. Otherwise the node will be created with primary node type nt:resource.

If the node to be created is nt:file, the actual file data will really be stored in the jcr:content child node of the new nt:file node whose primary node type is then set as nt:resource.

Example 1: Upload an image to a node named image below /content/page:

<**form** method="POST" action="/content/page" enctype="multipart/form-data">

<**input** type="file" name="image" />

<**input** type="Submit" />

</**form**>

Example 2: Upload a file as a node of type nt:file below /content/folder:

<**form** method="POST" action="/content/page" enctype="multipart/form-data">

<**input** type="file" name="\*" />

<**input** type="hidden" name="\*@TypeHint" value="nt:file" />

<**input** type="Submit" />

</**form**>

Assuming the user selected a file named myImage.jpg the uploaded file would be stored in an nt:file node at /content/folder/myImage.jpg.

Notes:

* Files are generally uploaded using <input type=’file’ > tag.
* File are not stored as property , instead new node is created for file with 3 properties (jcr:data =actual file content, jcr:lastModified, and jcr:mimeType)
* The name of the file node is taken from the name property of file tag. If the name is \* then actual file name is used to create the node.
* **@TypeHint** we can change the type of file node.
* If we are not providing the file node type in that case the node will be created as **nt: file** if it is under nt: folder. Otherwise it will be created as **nt:resource**.
* If node type is nt:file then the actual file will be stored in  jcr:content child node of the new nt:file node whose primary node type is then set as nt:resource.

##### **[Omitting Some Parameters](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "omitting-some-parameters)**

There may be times that you have forms which contain a lot of fields which you do not want to actually store in content. Such forms usually are created using some client-side GUI library which uses the fields for its own purposes. To be able to easily differentiate between real content to be actually stored and such control parameters, you may prefix the names of the fields destined for content with a dot-slash (./).

As soon as the SlingPostServlet encounters parameters prefixed with dot-slash, only those parameters are considered for content updates while all other parameters not prefixed are just ignored. In addition to dot-slash prefixed parameters, also parameters prefixed with dot-dot-slash (../) and slash (/) are considered in this situation.

For example, the following form only uses the first two fields for content update and ignores the rest:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="text" name="./title" />

<**input** type="text" name="../first/text" />

<**input** type="hidden" name="control0" /><!-- ignored -->

<**input** type="hidden" name="control1" /><!-- ignored -->

<**input** type="Submit" />

</**form**>

Because the SlingPostServlet encounters the ./title parameter, only parameters prefixed with dot-slash, dot-dot-slash and slash are considered for content update. In this case this would ./title and ../first/text while control0 and control1 are not prefixed and thus ignored.

Background: The name of the parameters used for content update are actually intended to be relative path names of the properties to modify. So in effect using the field name textis equivalent to ./text -- dot-slash meaning relative to the current node identified by the action attribute value for form tag -- or ../first/text if first is the name of the node to modify -- dot-dot-slash meaning relative to the parent node of the node identified by the action attribute value of the form tag.

In addition to the mechanism explained here, the following parameters are also ignored:

* Parameters whose name start with a colon (:) are always ignored by the SlingPostServlet with respect to content update. The reason is that the prefixing colon is intended as a marker for SlingPostServlet control parameters.
* The charset request parameter is also never written back because this parameter is used to convey the character encoding used to transport the request parameters.
* Request parameters matching a regular expression supplied with the servlet.post.ignorePattern configuration parameter are also ignored. By default this pattern is j\_.\*thus ignoring any request parameters with the prefix j\_ such as j\_username. Those request parameters are generally used for authentication purposes and may hit the Sling POST Servlet in some situations.

**Note:**

* In case we want to ignore few parameters for content update, we can mark the parameters in such a way that only those parameters will be allowed by SlingPostServlet.
* For that we have to prefix the parameter name with (./ or ../ or /).
* As soon as SlingPostServlet finds ./ prefixed parameter it starts ignoring any other parameter which are not are not prefixed with (./ or ../ or /).
* Parameter prefixed with : are always ignored as these are used for control parameters. Example :operation , :applyTo etc.
* Parameter name title is equivalent to ./title . The ./ prefix means relative to the parent.
* If the action="/content/page/first" and the parameter name is ../first/text , Then it is equivalent to parameter name text.

###### **@ suffixes parameters**

* @ suffixed parameters are not used on their own but always in conjunction with a plain parameter.
* Suffixed parameters can be used to do multiple things. Such as providing default value to some parameter, providing the datatype to the property.
* Ex @TypeHint, @ValueFrom, @CopyFrom,@MoveFrom, @DefaultValue, @UseDefaultWhenMissing,@IgnoreBlanks, @Delete,@Patch

###### **@TypeHint**

Parameters with the @TypeHint suffix may be used to force storing the named parameter in a property with the given type. The value of the @TypeHint parameter, if applied to a parameter for a property, is the JCR property type name. If the @TypeHint parameter is applied to a field upload parameter, the value is used to indicate the JCR primary node type for the node into which the uploaded file is stored.

If the @TypeHint value ends with [], it indicates a multi-value property. A multi-value property is usually auto-detected if there are multiple values for the property (i.e. request parameter). But if only a single value is present in the request, the desired property type needs to be explicitly defined as multi-value by stating @TypeHint=<type>[].

Example: The following form sets the numeric width, the boolean checked, and the multi-valued hobbys (with 3 values to enter) properties:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="text" name="width" />

<**input** type="hidden" name="width@TypeHint" value="Long" />

<**input** type="checkbox" name="checked" />

<**input** type="hidden" name="checked@TypeHint" value="Boolean" />

<**input** type="text" name="hobbys"/>

<**input** type="text" name="hobbys"/>

<**input** type="text" name="hobbys"/>

<**input** type="hidden" name="hobbys@TypeHint" value="String[]" />

<**input** type="Submit" />

</**form**>

In real applications you would need some JavaScript that allows to add/remove values, ie. add/remove inputs with the name "hobbys". Or a pure JavaScript based form post would be used, that gathers the properties to update programmatically, but the additional parameter hobbys@TypeHint=String[] would be the same.

The @TypeHint suffixed parameter is assumed to be single-valued. If the parameter has multiple values, only the first is actually used.

For multi-value properties, see also the @Patch option.

For more information on applying @TypeHint to a file upload parameter see the section on File Uploads above.

**Note:**

* @TypeHint when used for properties it represents the type of data it’s storing in the property. When it is used for field upload parameter then it represents the jcr:primaryType of the node.
* name="hobbys@TypeHint" value="String[ ]" [ ] is used along with type to represent multi-value property.

###### **@Delete**

Sometimes it may be required to not set a property to a specific value but to just remove it while processing the content update request. One such situation is a property filled from one or more checkboxes in an HTML form. If none of the checkboxes are checked, no parameter is actually submitted for these checkboxes. Hence the SlingPostServlet will not touch this property and effectively leave it untouched, while the natural reaction would have been to remove the property.

Here comes the @Delete suffixed parameter. This simply causes the indicated property be removed if it exists. If the property does not exist, nothing more happens. The actual value of the @Delete suffixed parameter does not care as long as the parameter is submitted.

Example: To ensure the color property is actually removed if no color has been selected, you might use the following form:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="checkbox" name="color" value="red" />

<**input** type="checkbox" name="color" value="green" />

<**input** type="checkbox" name="color" value="blue" />

<**input** type="hidden" name="color@Delete" value="delete text" /><!-- actual value is ignored -->

<**input** type="Submit" />

</**form**>

The @Delete suffixed parameter is also special in that there need not be a correlated parameter without a suffix. If both -- a parameters text and text@Delete are set, the textproperty is first deleted and then filled with the new content.

The @Delete suffixed parameter in fact calls for a sub-operation, which is executed after the node addressed by the request URL is created (if needed) but before any other tasks of content creation and modification are done. Any item -- this may be a property or a node, actually -- addressed by the @Delete suffixed parameter is just removed if it exists. If the item does not exist, nothing happens.

###### **@DefaultValue**

The @DefaultValue suffixed parameter may be provided to set a property to a default value should no value be provided in the actual parameters. Same as for normal parameters, the @DefaultValue parameter may have multiple values to create multi-valued properties.

Example: Set the text property to a default value if the user does not provide one:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="text" name="text" />

<**input** type="hidden" name="text@DefaultValue" value="--- Default Value ---" />

<**input** type="Submit" />

</**form**>

###### **@UseDefaultWhenMissing**

As described above, @DefaultValue only takes effect if no value is provided for a particular parameter. However, in some cases, such as HTML checkboxes, this isn't sufficient because the parameter isn't submitted at all. To handle this scenario, you can use the @UseDefaultWhenMissing suffixed parameter.

<form method="POST" action="/content/page/first" enctype="multipart/form-data">

<input name="queryIgnoreNoise" **class**="input" type="checkbox" **value**="true"/>

<input type="hidden" name="queryIgnoreNoise@DefaultValue" **value**="false"/>

<input type="hidden" name="queryIgnoreNoise@UseDefaultWhenMissing" **value**="true"/>

</form>

###### **@IgnoreBlanks**

Sometimes a form client will supply empty parameter values resulting in content being created or modified. For example submitting this form:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="hidden" name="stringProperty@TypeHint" value="String[]"/>

<**input** type="text" name="stringProperty" value="foo"/>

<**input** type="text" name="stringProperty" value="bar"/>

<**input** type="text" name="stringProperty" value=""/>

</**form**>

will result in multi-value String property being set to [ "foo", "bar", "" ]. Notice the blank value.

Likewise submitting this form without a value entered:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="hidden" name="stringProperty@TypeHint" value="String"/>

<**input** type="text" name="stringProperty" value=""/>

</**form**>

will result in the single-value String property being set to an empty string.

To overcome this situation the @IgnoreBlanks suffix may be used to consider parameters with an empty string value to be ignored during processing. That is such parameter values would be treated as if they would not be supplied.

Adding

<input type="hidden" name="stringProperty@IgnoreBlanks" value="true"/>

to the above forms will cause the multi-value property be set to the two-element value [ "foo", "bar" ] and to not modify the property at all in the second single-value example.

###### [**@ValueFrom**](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html#valuefrom)

In some situations, an HTML form with parameters may be reused to update content. But one or more form parameters may not comply with the names expected to be used for properties. In this case a parameter suffixed with @ValueFrom may be set containing the name of the parameter providing the actual data to be used.

Example: To set the property text from a form element supplied\_text, you might use the following form:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="text" name="supplied\_text" />

<**input** type="hidden" name="./text@ValueFrom" value="supplied\_text" />

<**input** type="Submit" />

</**form**>

To prevent storing the additional parameters in the repository you might want to use the prefixing mechanism as shown in the example above, where the @ValueFrom parameter is prefixed and thus the supplied\_text parameter is not used for property setting.

The @ValueFrom suffixed parameter is assumed to be single-valued. If the parameter has multiple values it is ignored completely.

The @ValueFrom suffixed parameter is also special in that there must not be a correlated parameter without a suffix. Thus have parameters text and text@ValueFrom may have unexpected results.

<form action=’/mynode’ method=POST>

<input type=’text’ name=’oldtitle’ />

<input type=’hidden’ name=’newtitle@ValueFrom’ value=’oldtitle’ />

</form>

Get the value from oldtitle and store with name as newtitle.

In this case the oldtitle will not be stored as property.

###### **@CopyFrom**

Similar to the @MoveFrom suffix exists a @CopyFrom suffix. The latter works exactly the same as the former except that the item addressed by the parameter value is not moved but just copied.

Example: Your Flash-based file upload stored the file on the server at /tmp/upload/123. You now want to store this file along with a title and a text in a newly created node. The following form may be your friend:

<!-- trailing slash generates a name for the new node -->

<**form** method="POST" action="/content/page/" enctype="multipart/form-data">

<**input** type="hidden" name="image@CopyFrom" value="/tmp/upload/123" />

<**input** type="text" name="title" />

<**input** type="text" name="text" />

<**input** type="Submit" />

</**form**>

If there exists no repository item at the indicated path, nothing is done. If the item indicated by the @CopyFrom suffixed parameter already exists, it is replaced by the item addressed by the parameter value -- unless of course there is no item at the named location.

The @CopyFrom suffixed parameter is assumed to be single-valued. If the parameter has multiple values it is ignored completely.

The @CopyFrom suffixed parameter is also special in that there must not be a correlated parameter without a suffix. Thus have parameters text and text@CopyFrom may have unexpected results.

The @CopyFrom suffixed parameter in fact calls for a sub-operation, which is executed after the @MoveFrom sub operation but before any other tasks of content creation and modification are done.

@CopyFrom can also be used to copy a particular property from a different node.

<Input type=’hidden’ name=’title@CopyFrom’ value=’/node/prop’ >

###### **@MoveFrom**

Now, that your bright and shiny content management application has great Flash-based file upload feature you will want to be able to use the pre-uploaded files for your content with the same request as when you upload other content. For example you might have a node storing some text and an illustration you uploaded as an image file.

To support this kind of functionality, the @MoveFrom suffixed parameter may be set to the repository path of the node to where you uploaded the image file.

Example: Your Flash-based file upload stored the file on the server at /tmp/upload/123. You now want to store this file along with a title and a text in a newly created node. The following form will be your friend:

<!-- trailing slash generates a name for the new node -->

<**form** method="POST" action="/content/page/" enctype="multipart/form-data">

<**input** type="hidden" name="image@MoveFrom" value="/tmp/upload/123" />

<**input** type="text" name="title" />

<**input** type="text" name="text" />

<**input** type="Submit" />

</**form**>

If there exists no repository item at the indicated path, nothing is done. If the item indicated by the @MoveFrom suffixed parameter already exists, it is replaced by the item addressed by the parameter value -- unless of course there is no item at the named location.

The @MoveFrom suffixed parameter is assumed to be single-valued. If the parameter has multiple values it is ignored completely.

The @MoveFrom suffixed parameter is also special in that there must not be a correlated parameter without a suffix. Thus have parameters text and text@MoveFrom may have unexpected results.

The @MoveFrom suffixed parameter in fact calls for a sub-operation, which is executed after the @Delete sub operation but before any other tasks of content creation and modification are done.

###### **@Patch**

When modifying multi-value properties, the @Patch suffix can be used to just add + or remove - individual values without overwriting the full array. This allows to change the array without knowing the current values.

For example, imagine a multi-value string property that stores tags or keywords. To both add a tag "cool" and remove "boring" from the list:

<**form** method="POST" action="/content/page/first" enctype="multipart/form-data">

<**input** type="hidden" name="tags@TypeHint" value="String[]" />

<**input** type="hidden" name="tags@Patch" value="true" />

<**input** type="text" name="tags" value="+cool"/>

<**input** type="text" name="tags" value="-boring"/>

<**input** type="Submit" />

</**form**>

The array will be treated like a set: when adding a value, it will only be added once if it does not exist yet; when removing a value, all occurrences of it will be removed. For values not affected by the add or remove operations, nothing changes. An existing array with duplicate entries will not automatically be converted into a set.

The format for an individual parameter value is <operation><value>. If there is no or no valid operation given, this value will be ignored.

Operation + will add the <value> to the array if it is not part of it yet.

Operation - will remove all occurrences of <value> from the array.

The value of the @Patch suffixed parameter is irrelevant, it can be empty (example above uses true for clarity).

All types should be supported via @TypeHint, but it needs to indicate a multi-value property, ending with [].

##### [**Response Status**](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html#response-status)

The modification operation has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | An existing node has been updated with content |
| 201/CREATED | A new node has been created and filled with content |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request |

### [Content Removal](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "content-removal)

To remove existing content just address the item to be removed and set the :operation parameter to delete. For example the following command line removes the /content/sample page:

$ curl -F":operation=delete" http://host/content/sample

##### [**Response Status**](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html#response-status)

The delete operation has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | The resource (and all its descendants) has been removed |
| 404/NOT FOUND | The request URL does not address an existing repository item |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request |

##### **[Deleting Multiple Items](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "deleting-multiple-items)**

By using the :applyTo request parameter it is possible to remove multiple items in one single request. Deleting items in this way leaves you with less control, though. In addition, if a single item removal fails, no item at all is removed.

When specifying the item(s) to be removed with the :applyTo parameter, the request resource is left untouched (unless of course if listed in the :applyTo parameter) and only used to resolve any relative paths in the :applyTo parameter.

To remove the /content/page1 and /content/page2 nodes, for example, you might use the following command line:

$ curl -F":operation=**delete**" -F":applyTo=/**content**/page1" \

-F":applyTo=/**content**/page2" http://host/content/sample

Using a trailing star in the :applyTo parameter (as mentioned before), you can remove all the children of the /content node, for example, as follows:

$ curl -F":operation=delete" -F":applyTo=/content/\*" http://host/content/sample

If any resource listed in the :applyTo parameter does not exist, it is silently ignored.

###### [**Response Status**](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html#response-status)

The delete operation applied to multiple resources has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | All requested and existing resources have been removed |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request |

### [Copying Content](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "copying-content)

To copy existing content to a new location, the copy operation is specified. This operation copies the item addressed by the request URL to a new location indicated by the :destparameter. The :dest parameter is the absolute or relative path to which the resource is copied. If the path is relative it is assumed to be below the same parent as the request resource. If it is terminated with a / character the request resource is copied to an item of the same name under the destination path.

To illustrate the :dest parameter handling, lets look at a few examples. All examples are based on addressing the /content/sample item:

| **:dest Parameter** | **Destination Absolute Path** |
| --- | --- |
| /content/newSample | /content/newSample |
| different/newSample | /content/different/newSample |
| /content/different/ | /content/different/sample |
| different/ | /content/different/sample |

If an item already exists at the location derived from the :dest parameter, the copy operation fails unless the :replace parameter is set to true (case is ignored when checking the parameter value).

##### **Response Status**

The copy operation has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | The node has been copied to the new location replacing an existing item at the destination |
| 201/CREATED | The node has been copied to the new location creating a new item at the destination |
| 404/NOT FOUND | The request URL does not address an existing repository item |
| 412/PRECONDITION FAILED | An item already exists at the destination and the :replace parameter is not set to true |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request |

##### **Copying Multiple Items**

By using the :applyTo request parameter it is possible to copy multiple items in one single request. Copying items in this way leaves you with less control, though. In addition, if a single item copy fails, no item at all is copied.

When specifying the item(s) to be copied with the :applyTo parameter, the request resource is left untouched (unless of course if listed in the :applyTo parameter) and only used to resolve any relative paths in the :applyTo parameter.

To copy the /content/page1 and /content/page2 nodes to /content/target, for example, use:

$ curl -F":operation=copy" -F":applyTo=/content/page1" -F":applyTo=/content/page2" \

-F":dest=/content/target/" http://host/content/sample

Please note the trailing slash character (/) in the value of the :dest parameter. This is required for multi-item copy operations using the :applyTo parameter. The copied items are created below the node indicated by the :dest.

Using a trailing star in the :applyTo parameter (as mentioned before), you can copy all the children of the /content node, for example, as follows:

$ curl -F":operation=copy" -F":applyTo=/content/\*" -F":dest=/content/target/" \

http://host/content/sample

If any resource listed in the :applyTo parameter does not exist, it is silently ignored. Any item already existing at the copy destination whose name is the same as the name of an item to be copied is silently overwritten with the source item.

###### **Response Status**

The copy operation applied to multiple resources has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | All requested and existing resources have been copied |
| 412/PRECONDITION FAILED | The node indicated by the :dest parameter does not exist |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request. This status is also set if the :dest parameter value does not have a trailing slash character. |

### [Moving Content](https://sling.apache.org/documentation/bundles/manipulating-content-the-slingpostservlet-servlets-post.html" \l "moving-content)

To move existing content to a new location, the move operation is specified. This operation moves the item addressed by the request URL to a new location indicated by the :destparameter. The :dest parameter is the absolute or relative path to which the resource is moved. If the path is relative it is assumed to be below the same parent as the request resource. If it is terminated with a / character the request resource is moved to an item of the same name under the destination path.

To illustrate the :dest parameter handling, lets look at a few examples. All examples are based on addressing the /content/sample item:

| **:dest Parameter** | **Destination Absolute Path** |
| --- | --- |
| /content/newSample | /content/newSample |
| different/newSample | /content/different/newSample |
| /content/different/ | /content/different/sample |
| different/ | /content/different/sample |

If an item already exists at the location derived from the :dest parameter, the move operation fails unless the :replace parameter is set to true (case is ignored when checking the parameter value).

##### **Response Status**

The move operation has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | The node has been moved to the new location replacing an existing item at the destination |
| 201/CREATED | The node has been moved to the new location creating a new item at the destination |
| 404/NOT FOUND | The request URL does not address an existing repository item |
| 412/PRECONDITION FAILED | An item already exists at the destination and the :replace parameter is not set to true |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request |

##### **Moving Multiple Items**

By using the :applyTo request parameter it is possible to move multiple items in one single request. Moving items in this way leaves you with less control, though. In addition, if a single item move fails, no item at all is moved.

When specifying the item(s) to be moved with the :applyTo parameter, the request resource is left untouched (unless of course if listed in the :applyTo parameter) and only used to resolve any relative paths in the :applyTo parameter.

To for example move the /content/page1 and /content/page2 nodes to /content/target, you might use the following command line:

$ curl -F":operation=move" -F":applyTo=/content/page1" -F":applyTo=/content/page2" \

-F":dest=/content/target/" http://host/content/sample

Please note the trailing slash character (/) in the value of the :dest parameter. This is required for multi-item move operations using the :applyTo parameter. The moved items are created below the node indicated by the :dest.

Using a trailing star in the :applyTo parameter (as mentioned before), you can move all the children of the /content node, for example, as follows:

$ curl -F":operation=move" -F":applyTo=/content/\*" -F":dest=/content/target/" \

http://host/content/sample

If any resource listed in the :applyTo parameter does not exist, it is silently ignored. Any item already existing at the move destination whose name is the same as the name of an item to be moved is silently overwritten with the source item.

###### **Response Status**

The move operation applied to multiple resources has the following status responses:

| **Status** | **Explanation** |
| --- | --- |
| 200/OK | All requested and existing resources have been moved |
| 412/PRECONDITION FAILED | The node indicated by the :dest parameter does not exist |
| 500/INTERNAL SERVER ERROR | Some exception, for example a RepositoryException, occurred while processing the request. This status is also set if the :dest parameter value does not have a trailing slash character. |

Sling post servlet Default Handler

<form action=’/mynode’ method=POST>

<input type=’text’ name=’title’ />

<textarea name=’body’ />

</form>

Create or update node mynode and set value of title and body property .also set **lastModified** and **lastModifiedBy** property.

<form action=’/mynode/’ method=POST>

<input type=’text’ name=’dummy’ />

<input type=’hidden’ name=’:order’ value=’first’/>

</form>

This will create a node below /mynode with name dummy.

And this will be the first child.

Other possible values of **:order** are **last** , **after** x, **before** x where x can be any number.

<form action=’/node’ method=POST>

<input type=’hidden’ name=’: operation’ value=’delete’/>

<input type=’hidden’ name=’:applyTo’ value=’/node/one’/>

<input type=’hidden’ name=’: applyTo’ value=’/node/two’/>

</form>

Used for deleting specific nodes under /node (here it will delete **/node/one** and **/node/two** node).

<form action=’/mynode/’ method=POST>

<input type=’hidden’ name=’:name’ value=’new\_node’/>

<input type=’hidden’ name=’: nameHint’ value=’new node’/>

</form>

Create node below /mynode and use **:name or : nameHint** property and set **created** and **createdBy**

<form action=’/mynode’ method=POST>

<input type=’text’ name=’customer’ />

<input type=’hidden’ name=’customer@DefaultValue’ value=’john Doe’ />

<input type=’hidden’ name=’title@Delete’ />

</form>

Setting up default value for customer property and removing the title property.

<form action=’/old/node’ method=POST>

<input type=’hidden’ name=’: operation’ value=’move’/>

<input type=’hidden’ name=’: dest’ value=’/new/place’/>

</form>

This will move the node from **/old/node**  to **/new/place**

<form action=’/mynode’ method=POST>

<input type=’text’ name=’date1’ value=’2008-06-13T18:55:00’ />

<input type=’text’ name=’date2’ />

<input type=’hidden’ name=’date2@TypeHint’ value=’Date’ />

<input type=’hidden’ name=’./uploaded/jcr:primaryType’ value=’nt:file’ />

</form>

Determining property type from date pattern in date1

Explicitly setting property type for date2.

Setting node type explicitly.

<form action=’/old/node’ method=POST>

<input type=’hidden’ name=’:operation’ value=’copy’ />

<input type=’hidden’ name=’:dest’ value=’/new/place’ />

<input type=’hidden’ name=’:replace’ value=’true’ />

</form>

It’s like swapping the nodes.

Debugging in aem

-Xrunjdwp:transport=dt\_socket,server=y,suspend=n,address=3000

Add above to crx-quickstart/bin start.bat file

::\* default JVM options

if not defined CQ\_JVM\_OPTS set CQ\_JVM\_OPTS=-Xmx1024m -XX:MaxPermSize=256M -Djava.awt.headless=true -Xrunjdwp:transport=dt\_socket,server=y,suspend=n,address=3000

Workflow