Components

What exactly is a Component?

Components:

- are modular units which realize specific functionality to present your content on your website
- are re-usable
- are developed as self-contained units within one folder of the repository
- have no hidden configuration files
- can contain other components
- can run anywhere within any AEM system. They can also be limited to run under specific components
- have a standardized user interface
- use widgets
- have an edit behaviour that can be configured

As components are modular, you can develop a new component on your local instance, then deploy this seamlessly to your test, then live environments.

Each AEM component:

- is a resource type
- is a collection of scripts that completely realize a specific function
- can function in "isolation"; this means either within AEM or a portal

Components included with AEM include:

- paragraph system
- header
- image, with accompanying text
- toolbar

NOTE

The functionality provided by Components and Widgets was implemented by the *cfc* libraries in CQ 4.

PROPERTIES AND CHILD NODES OF A COMPONENT

A component is a node of type cq:Component and has the following properties and child nodes:

Name	Type	Description
.	cq:Component	Current component. A component is of node type cq:Component.
allowedChildren	String[]	Path of a component that is allowed to be a child of this component.
allowedParents	String[]	Path of a component that is allowed to be a parent of this component.
componentGroup	String	Group under which the component can be selected in the Sidekick.
cq:cellName	String	If set, this property is taken as Cell Id. For more information, please refer to this article in the Knowledge Base.

cq:isContainer	Boolean	Checks if this component is a container component. For example a paragraph system component.
cq:noDecoration	Boolean	If true, the component is not rendered with automatically generated div and css classes.
cq:templatePath	String	Path to a node to use as a content template when the component is added from Sidekick. This must be an absolute path, not relative to the component node. Unless you want to reuse content already available elsewhere, this is not required and cq:template is sufficient (see below)
dialogPath	String	Path to a dialog in case that the component does not have a dialog node.
jcr:created	Date	Date of creation of the component.
jcr:description	String	Description of the component.
jcr:title	String	Title of the component.
sling:resourceSuperType	String	When set, the component inherits from this component.
<breadcrumb.jsp>	nt:file	Script file.
design_dialog	cq:Dialog	Definition of the design dialog.
cq:childEditConfig	cq:EditConfig	When the component is a container, as for example a paragraph system, this drives the edit configuration of the child nodes.
cq:editConfig	cq:EditConfig	Edit configuration of the component.
cq:htmlTag	nt:unstructured	Returns additional tag attributes that are added to the surrounding html tag. Enables addition of attributes to the automatically generated divs.
cq:template	nt:unstructured	If found, this node will be used as a content template when the component is added from Sidekick.

dialog	nt:unstructured	Definition of the edit dialog.
icon.png	nt:file	Icon of the component, appears next to the Title in Sidekick.
thumbnail.png	nt:file	Optional thumbnail that is shown while the component is dragged into place from Sidekick.
virtual	sling:Folder	Enables creation of virtual components. To see an example, please look at the contact component at /libs/foundation/components/profile/form/contact.

Default Components within AEM

AEM comes with a variety of out-of-the-box components that provide comprehensive functionality for website authors. These components and their usage within the installed "Geometrixx" website are a reference on how to implement and use components. The components are provided with all source code and can be used as is or as starting point for modified or extended components.

To get a list of all the components available in the repository, proceed as follows:
1. In CRXDE Lite, select **Tools** from the toolbar, then **Query**; the **Query** tab will be opened.
2. As **Type**, select XPath.
3. In the **Query** input field, enter following string:
   //element(*, cq:Component)
4. Click **Execute**. The list is displayed.

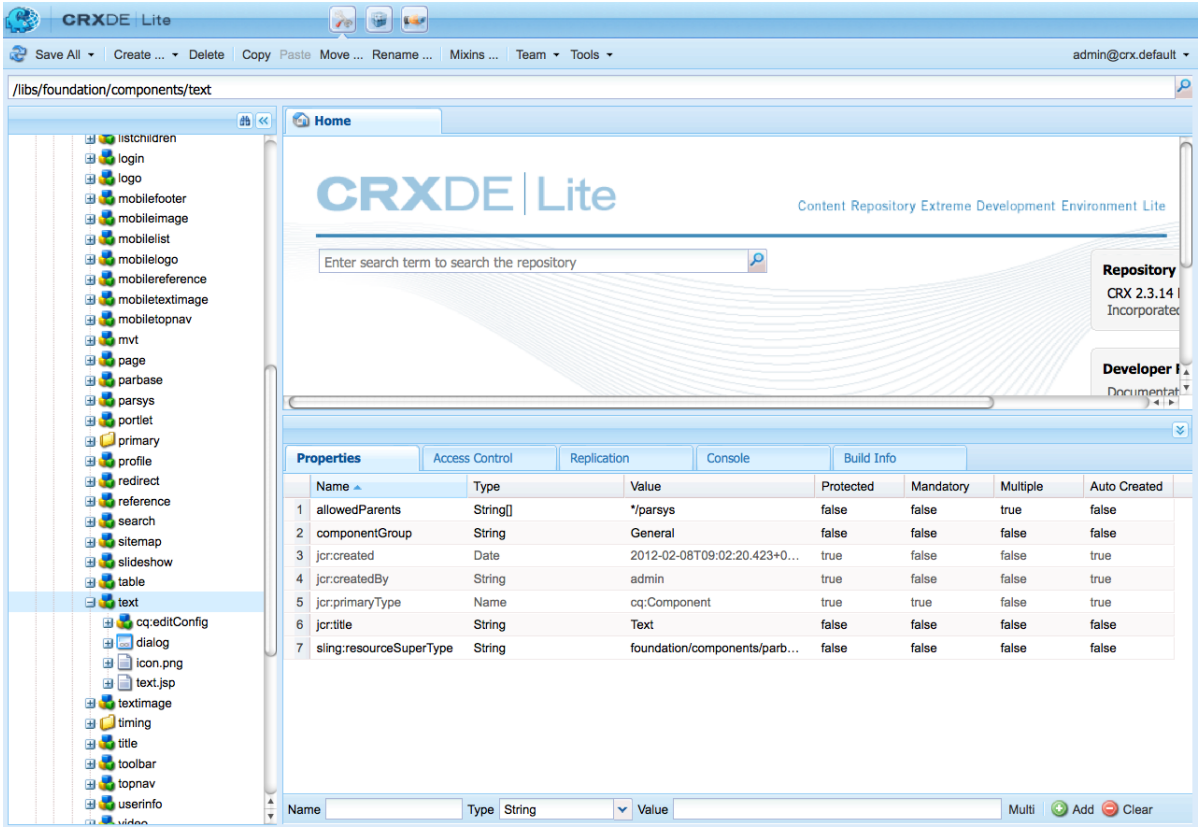
For detailed information on all the default components, including those that are not available out-of-the-box in AEM, see Default Components

Components and their structure

Component definitions

As already mentioned, the definition of a component can be broken down into:
- AEM components are based on Sling
- AEM components are located under /libs/foundation/components
- site specific components are located under /apps/<sitename>/components
- AEM has the *page* component; this is a particular type of resource which is important for content management.
- AEM standard components are defined as cq:Component and have the elements:
  - a list of jcr properties; these are variable and some may be optional though the basic structure of a component node, its properties and subnodes are defined by the cq:Component definition
  - the dialog defines the interface allowing the user to configure the component
  - the child node cq:editConfig (of type cq:EditConfig) defines the edit properties of the component and enables the component to appear in the Sidekick.
    Note: if the component has a dialog, it will automatically appear in the Sidekick, even if the cq:editConfig does not exist.
  - resources: define static elements used by the component
  - scripts are used to implement the behavior of the resulting instance of the component
  - thumbnail: image to be shown if this component is listed within the paragraph system

If we take the text component, we can see these elements:



Properties of particular interest include:

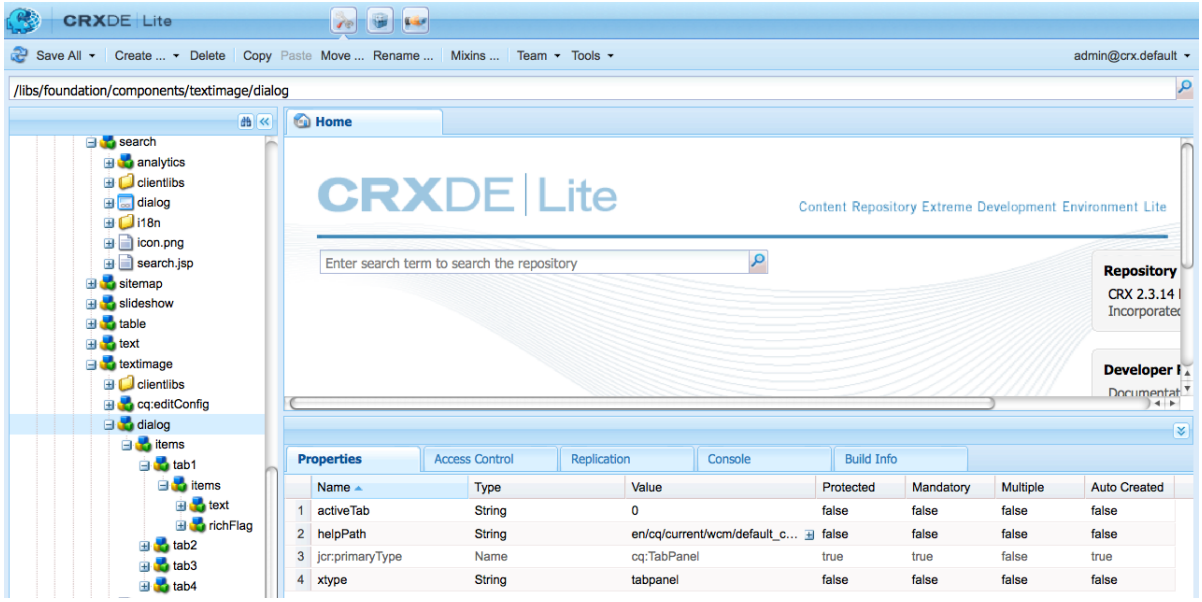
- jcr:title - title of the component; for example, appears in the component list within sidekick
- jcr:description - description for the component; again appears in the component list within sidekick
- allowedChildren - components which are allowed as children
- allowedParents - components which can be specified as parents
- icon.png - graphics file to be used as an icon for the component in Sidekick
- thumbnail.png - graphics file to be used as a thumbnail for the component while dragging it from Sidekick

Child nodes of particular interest include:

- cq:editConfig (cq:EditConfig) - this controls visual aspects; for example, it can define the appearance of a bar or widget, or can add customized controls
- cq:childEditConfig(cq:EditConfig) - this controls the visual aspects for child components that do not have their own definitions
- dialog (nt:unstructured) - defines the dialog for editing content of this component
- design_dialog (nt:unstructured) - specifies the design editing options for this component

DIALOGS

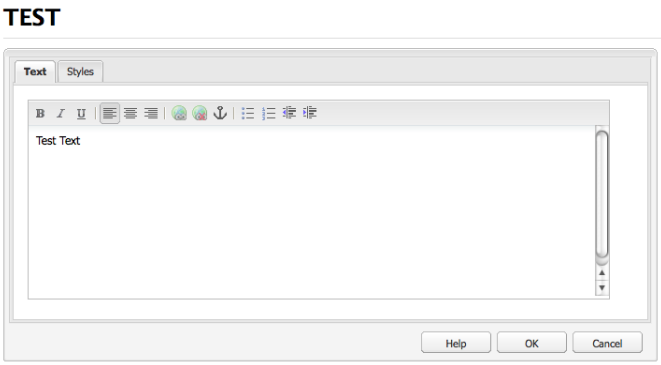
Dialogs are a key element of your component as they provide an interface for authors to configure and provide input to that component.
Depending on the complexity of the component your dialog may need one or more tabs - to keep the dialog short and to sort the input fields.
For example, you can create the dialog as cq:Dialog, which will provide a single tab - as in the text component, or if you need multiple tabs, as with the textimage component, the dialog can be defined as cq:TabPanel:



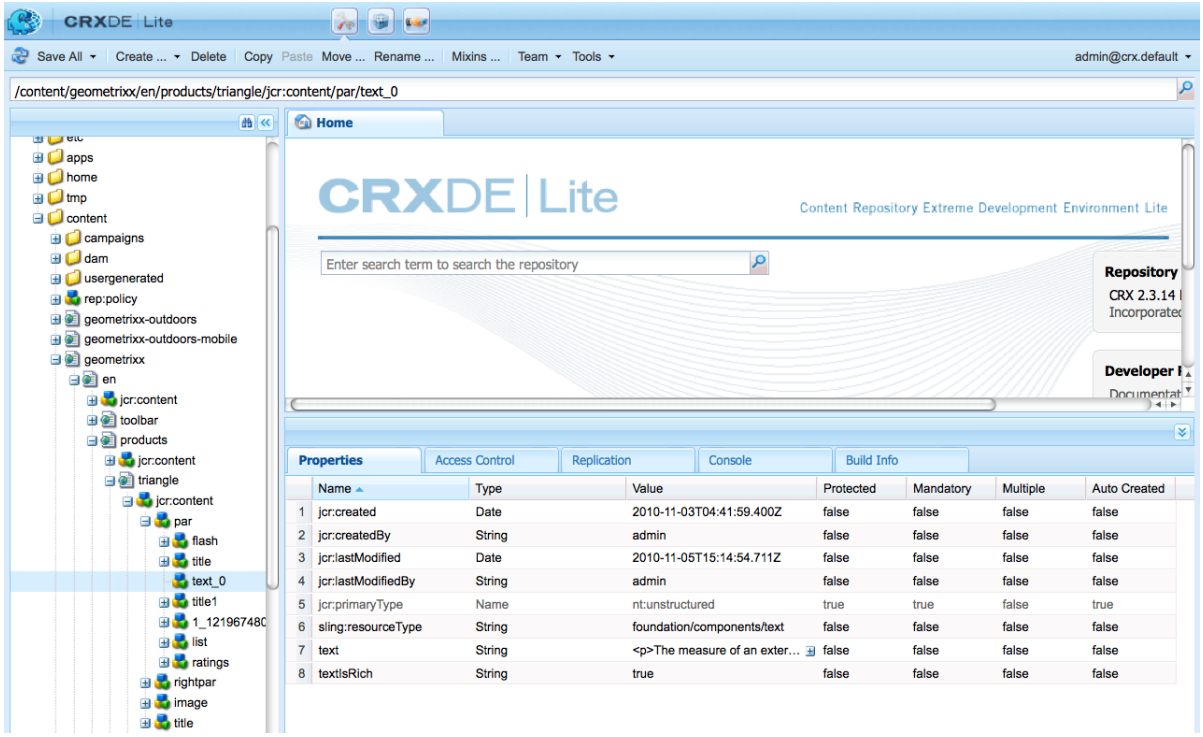
Within a dialog, a cq:WidgetCollection (items) is used to provide a base for either input fields (cq:Widget) or further tabs (cq:Widget). This hierarchy can be extended.

COMPONENT DEFINITIONS AND THE CONTENT THEY CREATE

If we create an instance of the Text paragraph (as above) on the <content-path>/Test.hml page:



then we can see the structure of the content created within the repository:



In particular, if you look at the text:

- the definition of /libs/foundation/components/text/dialog/items/tab1/items/text has the property name=./text
- within the content, this generates the property title holding the input Test Text.

The properties defined are dependent on the individual definitions, which although they can be more complex than above, follow the same basic principles.

COMPONENT HIERARCHY AND INHERITANCE

Components within AEM are subject to 3 different hierarchies:

- Resource Type Hierarchy:
  This is used to extend components using the property sling:resourceSuperType. This enables the component to inherit; for example a text component will inherit various attributes from the standard component.
  - scripts (resolved by Sling)
  - dialogs
  - descriptions (including thumbnail images, icons, etc)
- Container Hierarchy :
  This is used to populate configuration settings to the child component and is most commonly used in a parsys scenario.
  For example, configuration settings for the edit bar buttons, control set layout (editbars, rollover), dialog layout (inline, floating) can be defined on the parent component and propagated to the child components.
  Configuration settings (related to edit functionality) in cq:editConfig and cq:childEditConfig are propagated.
- Include Hierarchy
  This is imposed at runtime by the sequence of includes.
  This hierarchy is used by the Designer, which in turn acts as the base for various design aspects of the rendering; including layout information, css information, the available components in a parsys among others.

SUMMARY

The following summary applies for every component.
Summary:

- Location: /apps/<myApp>/components

Root Node:

- <mycomponent> (cq:Component) - Hierarchy node of the component.

Vital Properties:

- jcr:title - Component title; for example, used as a label when the component is listed within the sidekick.
- jcr:description - Component description; for example, also shown in the component list of the sidekick.
- allowedChildren - Specifies the allowed child components.
- allowedParents - Specifies the allowed parent components.
- icon.png - Icon for this component.

Vital Child Nodes:

- cq:editConfig (cq:EditConfig) - Controls author UI aspects; for example, bar or widget appearance, adds custom controls.
- cq:childEditConfig (cq:EditConfig) - Controls author UI aspects for child components that do not define their own cq:editConfig.
- dialog (cq:Dialog) - Content editing dialog for this component.
- design_dialog (cq:Dialog) - Design editing for this component.

Developing Components

This section describes how to create your own components and add them to the paragraph system.

A quick way to get started is to copy an existing component and then make the changes you want.

An example of how to develop a component is described in detail in Extending the Text and Image Component - An Example.

DEVELOPING A NEW COMPONENT BY ADAPTING AN EXISTING COMPONENT

To develop new components for AEM based on existing component you can copy the component, create a javascript file for the new component and store it in a location accessible to AEM (see also Customizing Components and Other Elements):

1. In the CRXDE Lite, create a new component folder in /apps/<*myProject*>/components/<*myComponent*> by copying an existing component, such as the Text component, and renaming it.
   For example, to customize the Text component copy:
   from /libs/foundation/components/text
   to /apps/myProject/components/text

   NOTE

   It is recommended that you save your tree before copying the new component. You can do this by pressing the **Save All** button in the upper left corner of the CRXDE Lite console.

2. Modify the jcr:title to reflect its new name. This is required in order to avoid confusion with the foundation component.

3. Open the new component folder and make the changes you require; also, delete any extraneous information in the folder.
   You can make changes such as:
   - adding a new field in the dialog box
   - replacing the .jsp file (name it after your new component)
   - or completely reworking the entire component if you want

   For example, if you take a copy of the standard Text component, you can add an additional field to the dialog box, then update the .jsp to process the input made there.
4. Navigate to the component and verify the value of the allowedParents property is */parsys, which makes it available to the paragraph system.
   Either cq:editConfig node, dialog, or design_dialog node should be present and properly initialized for the new component to appear.

5. Activate the new component in your paragraph system either by adding the value <path-to-component> (for example, /apps/geometrixx/components/myComponent) to the property components of the node /etc/designs/geometrixx/jcr:content/contentpage/par in CRX or by following the instructions in Adding new components to paragraph systems.
6. In AEM WCM, open a page in your web site and insert a new paragraph of the type you just created to make sure the component is working properly.

NOTE

To see timing statistics for page loading, you can use Ctrl-Shift-U - with ?debugClientLibs=true set in the URL.

ADDING A NEW COMPONENT TO THE PARAGRAPH SYSTEM (DESIGN MODE)

After the component has been developed, you add it to the paragraph system, which enables authors to select and use the component when editing a page.

1. Access a page within your authoring environment that uses the paragraph system; for example <contentPath>/Test.html.
2. Switch to Design mode by either:
   - adding ?wcmmode=design to the end of the URL and accessing again; for example: <contextPath>/ Test.html?wcmmode=design
   - clicking Design in Sidekick

   You are now in design mode and can edit the paragraph system.
3. Click Edit.
   A list of components belonging to the paragraph system are shown (all those defined with the property allowedParents=*/parsys). Your new component is also listed.
   The components can be activated (or deactivated) to determine which are offered to the author when editing a page.
4. Activate your component, then return to normal edit mode to confirm that it is available for use.

EXTENDING THE TEXT AND IMAGE COMPONENT - AN EXAMPLE

This section provides an example on how to extend the widely used text and image standard component with a configurable image placement feature.

The extension to the text and image component allows editors to use all the existing functionality of the component plus have an extra option to specify the placement of the image either:

- on the left-hand side of the text (current behavior and the new default)
- as well as on the right-hand side

After extending this component, you can configure the image placement through the component's dialog box.

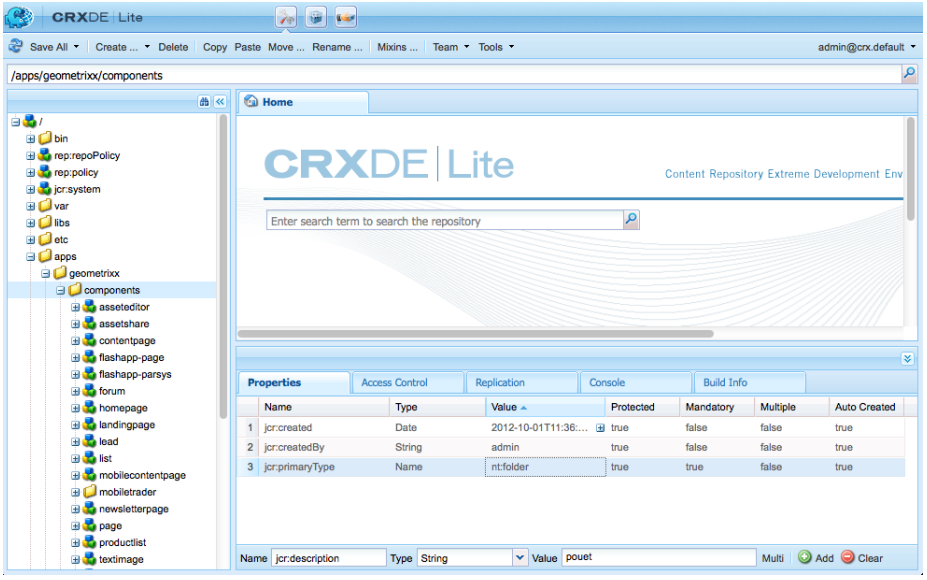
The following techniques are described in this exercise:

- Copying existing component node and modifying its metadata
- Modifying the component's dialog, including inheritance of widgets from parent dialog boxes
- Modifying the component's script to implement the new functionality

Extending the existing textimage component

To create the new component, we use the standard textimage component as a basis and modify it. We store the new component in the Geometrixx AEM WCM example application.

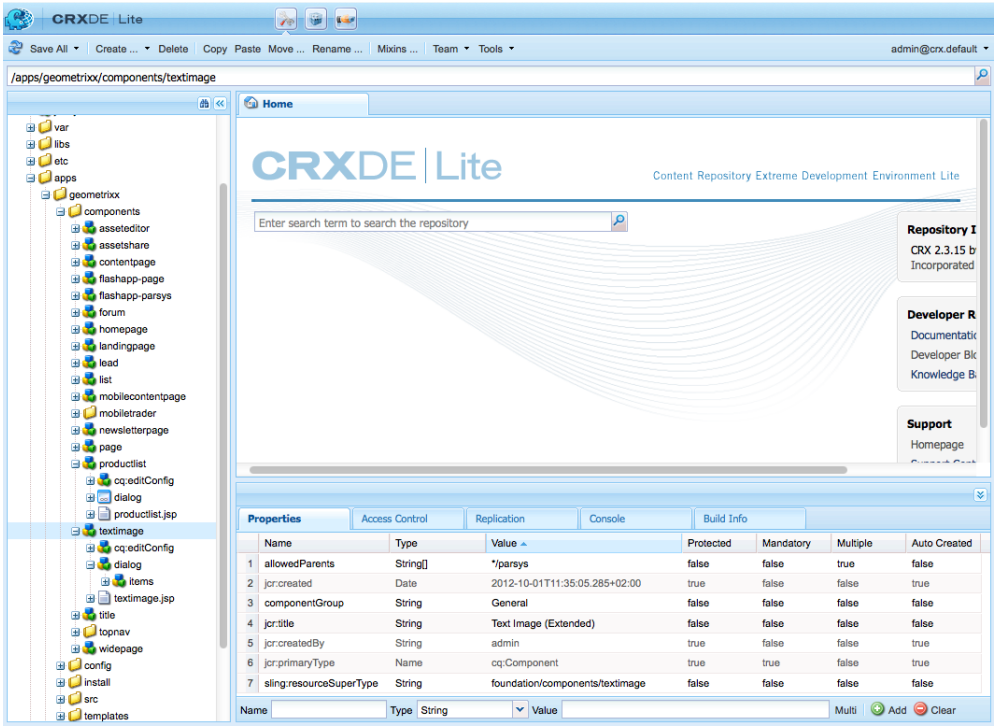
1. Copy the standard textimage component from /libs/foundation/components/textimage into the Geometrixx component folder, /apps/geometrixx/components, using textimage as the target node name. (Copy the component by navigating to the component, right-clicking and selecting Copy and browsing to the target directory.)



2. To keep this example simple, navigate to the component you copied and delete all the subnodes of the new textimage node except for the following ones:
   - dialog definition: textimage/dialog
   - component script: textimage/textimage.jsp
   - edit configuration node (allowing drag-and-drop of assets): textimage/cq:editConfig
3. Edit the component metadata:

   - Component name
     - Set jcr:description to Text Image Component (Extended)
     - Set jcr:title to Text Image (Extended)
   - Component listing in the paragraph (parsys component) system (leave as is)
     - Leave allowedParents defined as */parsys
   - Group, where the component is listed in the sidekick (leave as is)
     - Leave componentGroup set to General
   - Parent component for the new component (the standard textimage component)
     - Set sling:resourceSuperType to foundation/components/textimage

   After this step, the component node looks like this:



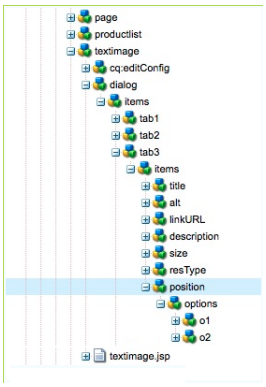
4. Change the sling:resourceType property of the edit configuration node of the image (property: textimage/cq:editConfig/cq:dropTargets/image/parameters/sling:resourceType) to geometrixx/components/textimage.

   This way, when an image is dropped to the component on the page, the sling:resourceType property of the extended textimage component is set to: geometrixx/components/textimage.
5. Modify the component's dialog box to include the new option. The new component inherits the parts of the dialog box that are the same as in the original. The only addition we make is to extend the **Advanced** tab, adding an **Image Position** dropdown list, with options **Left** and **Right**:

   - Leave the textimage/dialog properties unchanged.

   Note how textimage/dialog/items has four subnodes, tab1 to tab4, representing the four tabs of the textimage dialog box.
   - For the first two tabs (tab1 and tab2):
     - Change xtype to cqinclude (to inherit from the standard component).
     - Add a path property with values /libs/foundation/components/textimage/dialog/items/tab1.infinity.json and /libs/foundation/components/textimage/dialog/items/tab2.infinity.json, respectively.
     - Remove all other properties or subnodes.
   - For tab3:
     - Leave the properties and subnodes without changes
     - Add a new field definition to tab3/items, node position of type cq:Widget
     - Set the following properties (of type String) for the new tab3/items/position node:
       - name: ./imagePosition
       - xtype: selection
       - fieldLabel: Image Position
       - type: select
     - Add subnode position/options of type cq:WidgetCollection to represent the two choices for image placement, and under it create two nodes, o1 and o2 of type nt:unstructured.
     - For node position/options/o1 set the properties: text to Left and value to left.
     - For node position/options/o2 set the properties: text to Right and value to right.
   - Delete tab4.

   Image position is persisted in content as the imagePosition property of the node representing textimage paragraph. After these steps, the component dialog box looks like this:



NOTE

For more detail on the exact purpose of the nodes and properties referenced in this example, please refer to Components and their structure.

6. Extend the component script, textimage.jsp, with extra handling of the new parameter:

   1. Open the /apps/geometrixx/components/textimage/textimage.jsp script for editing.
   2. We are going to manipulate the style of the div tag, generated by the component, to float the image to the right. It is located in the following area of the code:

```
        image.addCssClass(ddClassName);
        image.setSelector(".img");
        image.setDoctype(Doctype.fromRequest(request));

        String divId = "cq-textimage-jsp-" + resource.getPath();
        String imageHeight = image.get(image.getItemName(Image.PN_HEIGHT));
```

   We are going to replace the emphasized code fragment *%> <%* with new code generating a custom style for this tag.

   3. Copy the following code fragment, and replace the *%><div class="image"* section of the line with it:

```
// todo: add new CSS class for the 'right image' instead of using
// the style attribute
String style="";
     if (properties.get("imagePosition", "left").equals("right")) {
          style = "style=\"float:right\"";
     }
     %><div <%= style %> class="image"><%
```

   Make sure that you leave the rest of the tag intact, otherwise the script will not function.
   For simplicity, we are hard-coding the style to the HTML tag. The proper way to do it would be to add a new CSS class to the application styles and just add the class to the tag in the code in the case of a right-aligned image.

   4. Save the component to the repository. The component is ready to test.

The code fragment, after the change, should look like this:

```
String ddClassName = DropTarget.CSS_CLASS_PREFIX + "image";

    if (image.hasContent() || WCMMode.fromRequest(request) == WCMMode.EDIT) {
        image.loadStyleData(currentStyle);
        // add design information if not default (i.e. for reference paras)
        if (!currentDesign.equals(resourceDesign)) {
            image.setSuffix(currentDesign.getId());
        }
        image.addCssClass(ddClassName);
        image.setSelector(".img");
```

```
        image.setDoctype(Doctype.fromRequest(request));

        String divId = "cq-textimage-jsp-" + resource.getPath();
        String imageHeight = image.get(image.getItemName(Image.PN_HEIGHT));
        // div around image for additional formatting
        String style="";
     if (properties.get("imagePosition", "left").equals("right")) {
          style = "style=\"float:right\"";
     }
     %><div <%= style %> class="image"><%
        %><% image.draw(out); %><br><%

        %><cq:text property="image/jcr:description" placeholder="" tagName="small" escapeXml="true"/
><%
        %></div>
        <%@include file="/libs/foundation/components/image/tracking-js.jsp"%>
        <%
    }

       String placeholder = (isAuthoringUIModeTouch && !image.hasContent())
               ? Placeholder.getDefaultPlaceholder(slingRequest, component, "", ddClassName)
               : "";
    %><cq:text property="text" tagClass="text" escapeXml="true" placeholder="<%= placeholder %>"/
><div
        class="clear"></div>
```

Checking the new component

After the component has been developed, you can add it to the paragraph system, which enables authors to select and use the component when editing a page. These steps allow you to test the component.

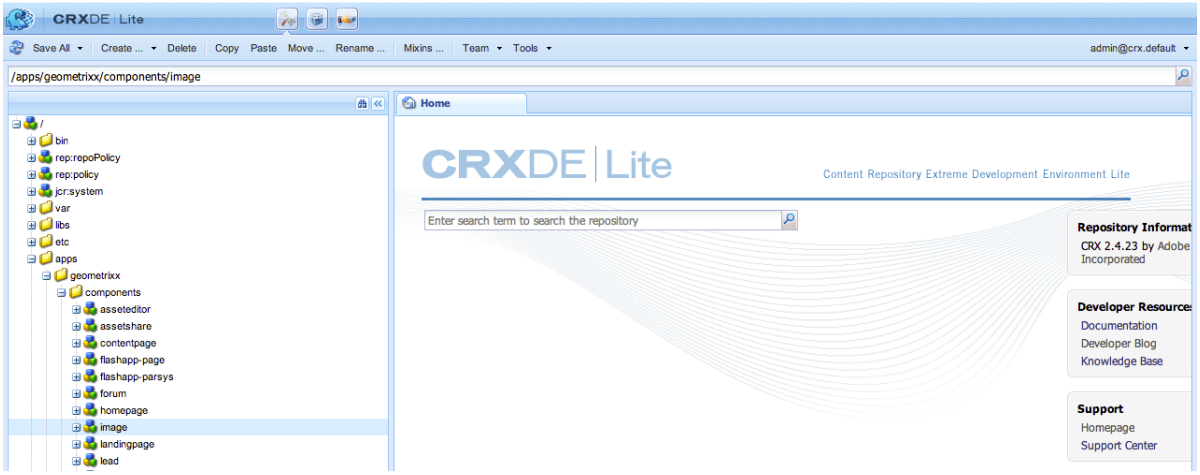
1. Open a page in Geometrixx; for example, English / Company.
2. Switch to design mode by clicking Design in Sidekick.
3. Edit the paragraph system design by clicking Edit on the paragraph system in the middle of the page. A list of components, which can be placed in the paragraph system are shown, and it should include your newly developed component, Text Image (Extended) . Activate it for the paragraph system by selecting it and clicking OK .
4. Switch back to the editing mode.
5. Add the Text Image (Extended) paragraph to the paragraph system, initialize text and image with sample content. Save the changes.
6. Open the dialog of the text and image paragraph, and change the Image Position on the Advanced tab to Right , and click OK to save the changes.
7. The paragraph is rendered with the image on the right.
8. The component is now ready to use.

The component stores its content in a paragraph on the Company page.

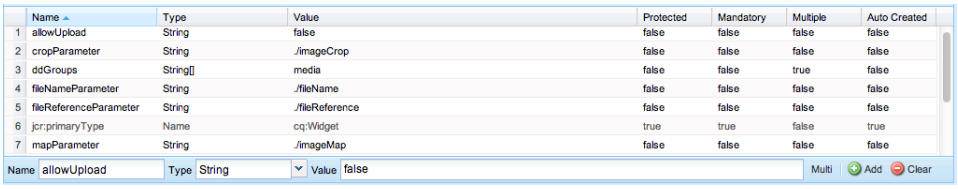
DISABLE UPLOAD CAPABILITY OF THE IMAGE COMPONENT

To disable this capability, we use the standard image component as a basis and modify it. We store the new component in the Geometrixx example application.

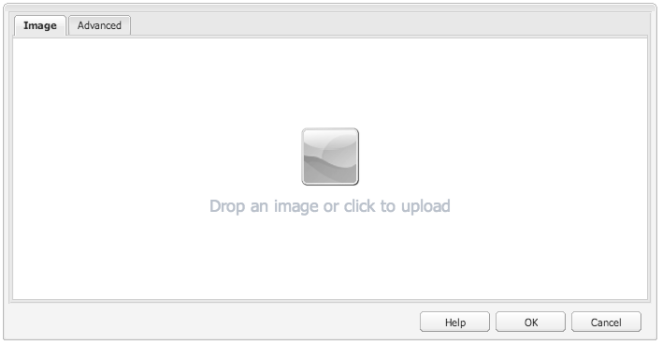
1. Copy the standard image component from /libs/foundation/components/image into the Geometrixx component folder, /apps/geometrixx/components, using image as the target node name.



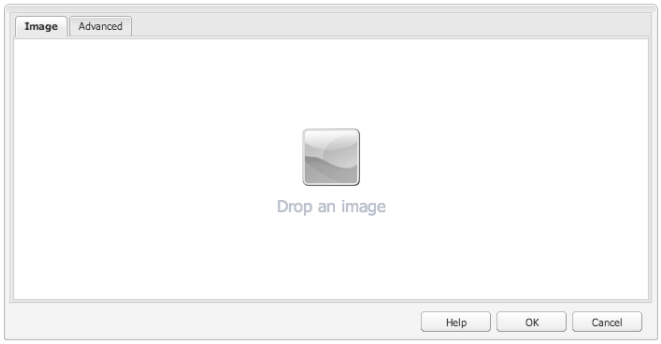
2. Edit the component metadata:
   - Set **jcr:title** to Image (Extended)
3. Navigate to /apps/geometrixx/components/image/dialog/items/image.
4. Add a new property:
   - **Name**: allowUpload
   - **Type**: String
   - **Value**: false



5. Click **Save All**. The component is ready to test.
6. Open a page in Geometrixx; for example, English / Company.
7. Switch to design mode and activate Image (Extended).
8. Switch back to the editing mode and add it to the paragraph system. On the next pictures, you can see the differences between the original image component and the one you just created.
   Original image component:



Your new image component:



9. The component is now ready to use.

MOVING COMPONENTS TO THE PUBLISH INSTANCE

The components that render content must be deployed on the same AEM instance as the content. Therefore, all components that are used for authoring and rendering pages on the author instance must be deployed on the publish instance. When deployed, the components are available to render activated pages. Use the following tools to move your components to the publish instance:

- Use Package Manager to add your components to a package and move them to another AEM instance.
- Use the Activate Tree replication tool to replicate the components.

Scripts

JSP Scripts or Servlets are usually used to render components.

According to the request processing rules of Sling the name for the default script is <componentname>.jsp.

global.jsp

The JSP script file global.jsp is used to provide quick access to specific objects (i.e. to access content) to any JSP script file used to render a component.
Therefore global.jsp should be included in every component rendering JSP script where one or more of the objects provided in global.jsp are used.

NOTE

Since CQ 5.3, the correct global.jsp path changed to /libs/foundation/global.jsp.
Note that the path /libs/wcm/global.jsp, which was used by earlier versions, is now obsolete.

FUNCTION OF GLOBAL.JSP, USED APIS AND TAGLIBS

The following lists the most important objects provided from the default global.jsp (/libs/foundation/global.jsp)
Summary:

- <cq:defineObjects />
  - slingRequest - The wrapped Request Object (SlingHttpServletRequest).
  - slingResponse - The wrapped Response Object (SlingHttpServletResponse).
  - resource - The Sling Resource Object (slingRequest.getResource();).
  - resourceResolver - The Sling Resource Resolver Object (slingRequest.getResoucreResolver();).
  - currentNode - The resolved JCR node for the request.
  - log - The Default logger ().
  - sling - The Sling script helper.
  - properties - The properties of the addressed resource (resource.adaptTo(ValueMap.class);).
  - pageProperties - The properties of the page of the addressed resource.
  - pageManager - The page manager for accessing AEM content pages (resourceResolver.adaptTo(PageManager.class);).

- component - The component object of the current AEM component..
- designer - The designer object for retrieving design information (resourceResolver.adaptTo(Designer.class);).
- currentDesign - The design of the addressed resource.
- currentStyle - The style of the addressed resource.

ACCESSING CONTENT

There are three methods to access content in AEM WCM:

- Via the properties object introduced in global.jsp:
  The properties object is an instance of a ValueMap (see Sling API) and contains all properties of the current resource.
  Example: String pageTitle = properties.get("jcr:title", "no title"); used in the rendering script of a page component.
  Example: String paragraphTitle = properties.get("jcr:title", "no title"); used in the rendering script of a standard paragraph component.
- Via the currentPage object introduced in global.jsp:
  The currentPage object is an instance of a page (see CQ5 API). The page class provides some methods to access content.
  Example: String pageTitle = currentPage.getTitle();
- Via currentNode object introduced in global.jsp:
  The currentNode object is an instance of a node (see JCR API). The properties of a node can be accessed by the getProperty() method.
  Example: String pageTitle = currentNode.getProperty("jcr:title");

JSP Tag Libraries

The CQ and Sling tag libraries give you access to specific functions for use in the JSP script of your templates and components.

CQ Tag Library

The CQ tag library contains helpful functions.

To use the CQ Tag Library in your script, the script must start with the following code:

```
<%@taglib prefix="cq" uri="http://www.day.com/taglibs/cq/1.0" %>
```

NOTE

When the /libs/foundation/global.jsp file is included in the script, the cq taglib is automatically declared.

When you develop the jsp script of a AEM component, it is recommended to include following code at the top of the script:

```
<%@include file="/libs/foundation/global.jsp"%>
```

It declares the sling, cq and jstl taglibs and exposes the regularly used scripting objects defined by the <cq:defineObjects /> tag. This shortens and simplifies the jsp code of your component.

<CQ:TEXT>

The <cq:text> tag is a convenience tag that outputs component text in a JSP.
It has the following optional attributes:

property

Name of the property to use. The name is relative to the current resource.

value

Value to use for output. If this attribute is present, it overwrites the use of the property attribute.

oldValue

Value to use for diff output. If this attribute is present, it overwrites the use of the property attribute.

escapeXml

Defines whether the characters <, >, &, ' and " in the resulting string should be converted to their corresponding character entity codes. Default value is false. Note that the escaping is applied after the optional formatting.

format

Optional java.text.Format to use for formatting the text.

noDiff

Suppresses the calculation of a diff output, even if a diff info is present.

tagClass

CSS class name of an element that will surround a non-empty output. If empty, no element is added.

tagName

Name of the element that will surround a non-empty output. It defaults to DIV.

placeholder

Default value to use for null or empty text in edit mode, i.e. the placeholder. Please note that the default check is performed after the optional formatting and escaping, i.e. it is written as-is to the output. It defaults to:
¶

default

Default value to use for null or empty text. Note that the default check is performed after the optional formatting and escaping i.e. it is written as-is to the output.

Some examples how the **<cq:text>** tag can be used in a JSP:

```
<cq:text property="jcr:title" tagName="h2"/>
<cq:text property="jcr:description" tagName="p"/>

<cq:text value="<%= listItem.getTitle() %>" tagName="h4" placeholder="" />
<cq:text value="<%= listItem.getDescription() %>" tagName="p" placeholder=""/>

<cq:text property="jcr:title" value="<%= title %>" tagName="h3"/><%
    } else if (type.equals("link")) {
        %><cq:text property="jcr:title" value="<%= "\u00bb " + title %>" tagName="p"
 tagClass="link"/><%
    } else if (type.equals("extralarge")) {
        %><cq:text property="jcr:title" value="<%= title %>" tagName="h1"/><%
    } else {
        %><cq:text property="jcr:title" value="<%= title %>" tagName="h2"/><%

<cq:text property="jcr:description" placeholder="" tagName="small"/>

<cq:text property="tableData"
               escapeXml="false"
               placeholder="<img src=\"/libs/cq/ui/resources/0.gif\" class=\"cq-table-placeholder\"
 alt=\"\">"
    />
```

```
<cq:text property="text"/>

<cq:text property="image/jcr:description" placeholder="" tagName="small"/>
<cq:text property="text" tagClass="text"/>
```

<CQ:SETCONTENTBUNDLE>

The <cq:setContentBundle> tag creates an i18n localization context and stores it in the javax.servlet.jsp.jstl.fmt.localizationContext configuration variable.
It has the following attributes:

language

The language of the locale for which to retrieve the resource bundle.

source

The source where the locale should be taken from. It can be set to one of the following values:

- **static**: the locale is taken from the language attribute if available, otherwise from the server default locale.
- **page**: the locale is taken from the language of the current page or resource if available, otherwise from the language attribute if available, otherwise from the server default locale.
- **request**: the locale is taken from the request locale (request.getLocale()).
- **auto**: the locale is taken from the language attribute if available, otherwise from the language of the current page or resource if available, otherwise from the request.

If the source attribute is not set (as it was the case until CQ 5.3):

- If the language attribute is set, the source attribute defaults to static.
- If the language attribute is not set, the source attribute defaults to auto.

<cq:setContentBundle>

The "content bundle" can be simply used by standard JSTL <fmt:message> tags. The lookup of messages by keys is two-fold:

1. First, the JCR properties of the underlying resource that is currently rendered are searched for translations. This allows you to define a simple component dialog to edit those values.
2. If the node does not contain a property named exactly like the key, the fallback is to load a resource bundle from the sling request (SlingHttpServletRequest.getResourceBundle(Locale)). The language or locale for this bundle is defined by the language and source attributes of the <cq:setContentBundle> tag.

The <cq:setContentBundle> tag can be used as follows in a jsp.
For pages that define their language:

```
... %><cq:setContentBundle source="page"/><%  %>
<div class="error"><fmt:message key="Hello"/>
</div> ...
```

For user personalized pages:

```
... %><cq:setContentBundle scope="request"/><% %>
<div class="error"><fmt:message key="Hello"/>
</div> ...
```

NOTE

The source attribute is new since CQ 5.4.

<CQ:INCLUDE>

The <cq:include> tag includes a resource into the current page.

It has the following attributes:

flush

- A boolean defining whether to flush the output before including the target.

path

- The path to the resource object to be included in the current request processing. If this path is relative it is appended to the path of the current resource whose script is including the given resource. Either path and resourceType, or script must be specified.

resourceType

- The resource type of the resource to be included. If the resource type is set, the path must be the exact path to a resource object: in this case, adding parameters, selectors and extensions to the path is not supported.
- If the resource to be included is specified with the path attribute that cannot be resolved to a resource, the tag may create a synthetic resource object out of the path and this resource type.
- Either path and resourceType, or script must be specified.

script

- The jsp script to include. Either path and resourceType, or script must be specified.

ignoreComponentHierarchy

- A boolean controlling whether the component hierarchy should be ignored for script resolution. If true, only the search paths are respected.

Example:

```
<%@taglib prefix="cq" uri="http://www.day.com/taglibs/cq/1.0" %><%
%><div class="center">
    <cq:include path="trail" resourceType="foundation/components/breadcrumb" />
    <cq:include path="title" resourceType="foundation/components/title" />
    <cq:include script="redirect.jsp"/>
    <cq:include path="par" resourceType="foundation/components/parsys" />
</div>
```

Should you use <%@ include file="myScript.jsp" %> or <cq:include script="myScript.jsp" %> to include a script?

- The <%@ include file="myScript.jsp" %> directive informs the JSP compiler to include a complete file into the current file. It is as if the contents of the included file were pasted directly into the original file.
- With the <cq:include script="myScript.jsp"> tag, the file is included at runtime.

Should you use <cq:include> or <sling:include>?

- When developing AEM components, Adobe recommends that you use <cq:include>.
- <cq:include> allows you to directly include script files by their name when using the script attribute. This takes component and resource type inheritance into account, and is often simpler than strict adherence to Sling's script resolution using selectors and extensions.

<CQ:INCLUDECLIENTLIB>

The <cq:includeClientLib> tag Includes a AEM html client library, which can be a js, a css or a theme library. For multiple inclusions of different types, for example js and css, this tag needs to be used multiple times in the jsp. This tag is a convenience wrapper around the com.day.cq.widget.HtmlLibraryManager service interface.
It has the following attributes:

categories

A list of comma-separated client lib categories. This will include all Javascript and CSS libraries for the given categories. The theme name is extracted from the request.

Equivalent to: com.day.cq.widget.HtmlLibraryManager#writeIncludes

theme

A list of comma-separated client lib categories. This will include all theme related libraries (both CSS and JS) for the given categories. The theme name is extracted from the request.

Equivalent to: com.day.cq.widget.HtmlLibraryManager#writeThemeInclude

js

A list of comma-separated client lib categories. This will include all Javascript libraries for the given categories.

Equivalent to: com.day.cq.widget.HtmlLibraryManager#writeJsInclude

css

A list of comma-separated client lib categories. This will include all CSS libraries for the given categories.

Equivalent to: com.day.cq.widget.HtmlLibraryManager#writeCssInclude

themed

A flag that indicates of only themed or non themed libraries should be included. If omitted, both sets are included. Only applies to pure JS or CSS includes (not for categories or theme includes).

The <cq:includeClientLib> tag can be used as follows in a jsp:

```
<%-- all: js + theme (theme-js + css) --%>
<cq:includeClientLib categories="cq.wcm.edit" />

<%-- only js libs --%>
<cq:includeClientLib js="cq.collab.calendar, cq.security" />

<%-- theme only (theme-js + css) --%>
<cq:includeClientLib theme="cq.collab.calendar, cq.security" />

<%-- css only --%>
<cq:includeClientLib css="cq.collab.calendar, cq.security" />
```

NOTE

The <cq:includeClientLib> tag is new since CQ 5.4.

<CQ:DEFINEOBJECTS>

The <cq:defineObjects> tag exposes the following, regularly used, scripting objects which can be referenced by the developer. It also exposes the objects defined by the <sling:defineObjects> tag.

componentContext

- the current component context object of the request (com.day.cq.wcm.api.components.ComponentContext interface).

component

- the current AEM component object of the current resource (com.day.cq.wcm.api.components.Component interface).

currentDesign

- the current design object of the current page (com.day.cq.wcm.api.designer.Design interface).

currentPage

- the current AEM WCM page object (com.day.cq.wcm.api.Page interface).

currentStyle

- the current style object of the current cell (com.day.cq.wcm.api.designer.Style interface).

designer

- the designer object used to access design information (com.day.cq.wcm.api.designer.Designer interface).

editContext

- the edit context object of the AEM component (com.day.cq.wcm.api.components.EditContext interface).

pageManager

- the page manager object for page level operations (com.day.cq.wcm.api.PageManager interface).

pageProperties

- the page properties object of the current page (org.apache.sling.api.resource.ValueMap).

properties

- the properties object of the current resource (org.apache.sling.api.resource.ValueMap).

resourceDesign

- the design object of the resource page (com.day.cq.wcm.api.designer.Design interface).

resourcePage

- the resource page object (com.day.cq.wcm.api.Page interface).
- It has the following attributes:

requestName

- inherited from sling

responseName

- inherited from sling

resourceName

- inherited from sling

nodeName

- inherited from sling

logName

- inherited from sling

resourceResolverName

- inherited from sling

slingName

- inherited from sling

componentContextName

- specific to wcm

editContextName

- specific to wcm

propertiesName

- specific to wcm

pageManagerName

- specific to wcm

currentPageName

- specific to wcm

resourcePageName

- specific to wcm

pagePropertiesName

- specific to wcm

componentName

- specific to wcm

designerName

- specific to wcm

currentDesignName

- specific to wcm

resourceDesignName

- specific to wcm

currentStyleName

- specific to wcm

Example

```
<%@page session="false" contentType="text/html; charset=utf-8" %><%
%><%@ page import="com.day.cq.wcm.api.WCMMode" %><%
%><%@taglib prefix="cq" uri="http://www.day.com/taglibs/cq/1.0" %><%
%><cq:defineObjects/>
```

NOTE

When the /libs/foundation/global.jsp file is included in the script, the <cq:defineObjects /> tag is automatically included.

<CQ:REQUESTURL>

The <cq:requestURL> tag writes the current request URL to the JspWriter. The two tags <cq:addParam> and <cq:removeParam> and may be used inside the body of this tag to modify the current request URL before it is written.
It allows you to create links to the current page with varying parameters. For example, it enables you to transform the request:
mypage.html?mode=view&query=something into mypage.html?query=something.
The use of addParam or removeParam only changes the occurrence of the given parameter, all other parameters are unaffected.
<cq:requestURL> does not have any attribute.
Examples:

```
<a href="<cq:requestURL><cq:removeParam name="language"/></cq:requestURL>">remove filter</a>
```

```
<a title="filter results" href="<cq:requestURL><cq:addParam name="language" value="${bucket.value}"/
></cq:requestURL>">${label} (${bucket.count})</a>
```

<CQ:ADDPARAM>

The <cq:addParam> tag adds a request parameter with the given name and value to the enclosing <cq:requestURL> tag.
It has the following attributes:
name

- name of the parameter to be added

value

- value of the parameter to be added

Example:

```
<a title="filter results" href="<cq:requestURL><cq:addParam name="language" value="${bucket.value}"/
></cq:requestURL>">${label} (${bucket.count})</a>
```

<CQ:REMOVEPARAM>

The <cq:removeParam> tag removes a request parameter with the given name and value from the enclosing <cq:requestURL> tag. If no value is provided all parameters with the given name are removed.
It has the following attributes:
name

- name of the parameter to be removed

Example:

```
<a href="<cq:requestURL><cq:removeParam name="language"/></cq:requestURL>">remove filter</a>
```

Sling Tag Library

The Sling tag library contains helpful Sling functions.

When you use the Sling Tag Library in your script, the script must start with the following code:

```
<%@ taglib prefix="sling" uri="http://sling.apache.org/taglibs/sling/1.0" %>
```

NOTE

When the /libs/foundation/global.jsp file is included in the script, the sling taglib is automatically declared.

<SLING:INCLUDE>

The <sling:include> tag includes a resource into the current page.
It has the following attributes:

flush

- A boolean defining whether to flush the output before including the target.

resource

- The resource object to be included in the current request processing. Either resource or path must be specified. If both are specified, the resource takes precedence.

path

- The path to the resource object to be included in the current request processing. If this path is relative it is appended to the path of the current resource whose script is including the given resource. Either resource or path must be specified. If both are specified, the resource takes precedence.

resourceType

- The resource type of the resource to be included. If the resource type is set, the path must be the exact path to a resource object: in this case, adding parameters, selectors and extensions to the path is not supported.
- If the resource to be included is specified with the path attribute that cannot be resolved to a resource, the tag may create a synthetic resource object out of the path and this resource type.

replaceSelectors

- When dispatching, the selectors are replaced with the value of this attribute.

addSelectors

- When dispatching, the value of this attribute is added to the selectors.

replaceSuffix

- When dispatching, the suffix is replaced by the value of this attribute.

NOTE

The resolution of the resource and the script that are included with the <sling:include> tag is the same as for a normal sling URL resolution. By default, the selectors, extension, etc. from the current request are used for the included script as well. They can be modified through the tag attributes: for example replaceSelectors="foo.bar" allows you to overwrite the selectors.

Examples:

```
<div class="item"><sling:include path="<%= pathtoinclude %>"/></div>
<sling:include resource="<%= par %>"/>
<sling:include addSelectors="spool"/>
<sling:include resource="<%= par %>" resourceType="<%= newType %>"/>
<sling:include resource="<%= par %>" resourceType="<%= newType %>"/>
<sling:include replaceSelectors="content" />
```

<SLING:DEFINEOBJECTS>

The <sling:defineObjects> tag exposes the following, regularly used, scripting objects which can be referenced by the developer:

slingRequest

- SlingHttpServletRequest object, providing access to the HTTP request header information - extends the standard HttpServletRequest - and provides access to Sling-specific things like resource, path info, selector, etc.

slingResponse

- SlingHttpServletResponse object, providing access for the HTTP response that is created by the server. This is currently the same as the HttpServletResponse from which it extends.**request**
- The standard JSP request object which is a pure HttpServletRequest.**response**
- The standard JSP response object which is a pure HttpServletResponse.

resourceResolver

- The current ResourceResolver object. It is the same as slingRequest.getResourceResolver()

.**sling**

- A SlingScriptHelper object, containing convenience methods for scripts, mainly sling.include('/some/other/resource') for including the responses of other resources inside this response (eg. embedding header html snippets) and sling.getService(foo.bar.Service.class) to retrieve OSGi services available in Sling (Class notation depending on scripting language).

resource

- the current Resource object to handle, depending on the URL of the request. It is the same as slingRequest.getResource().

currentNode

- If the current resource points to a JCR node (which is typically the case in Sling), this gives direct access to the Node object. Otherwise this object is not defined.

log

- Provides an SLF4J Logger for logging to the Sling log system from within scripts, eg. log.info("Executing my script").

- It has the following attributes:

requestName
responseName

nodeName
logName resourceResolverName
slingName

Example:

```
<%@page session="false" %><%
%><%@page import="com.day.cq.wcm.foundation.forms.ValidationHelper"%><%
%><%@taglib prefix="sling" uri="http://sling.apache.org/taglibs/sling/1.0" %><%
%><sling:defineObjects/>
```

JSTL Tag library

The JavaServer Pages Standard Tag Library contains a lot of useful and standard tags. The core, formatting and functions taglibs are defined by the /libs/foundation/global.jsp as shown in the following snippet.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

After importing the /libs/foundation/global.jsp file as described before, you can use the c, fmt and fn prefixes to access to those taglibs. The official documentation of the JSTL is available at The Java EE 5 Tutorial - JavaServer Pages Standard Tag Library.

Using Client-Side HTML Libraries

Modern websites rely heavily on client-side processing driven by complex JavaScript and CSS code. Organizing and optimizing the serving of this code can be a complicated issue.
To help deal with this issue, AEM provides **Client-side Library Folders**, which allow you to store your client-side code in the repository, organize it into categories, and define when and how each category of code is to be served to the client. The client-side library system then takes care of producing the correct links in your final webpage to load the correct code.
Read Using Client-Side HTML Libraries for more information.

Configuring the Edit Behaviour of a Component

This section explains how to configure the edit behaviour of a component.

The edit behaviour of a component is configured by adding a **cq:editConfig** node of type **cq:EditConfig** below the component node (of type **cq:Component**) and by adding specific properties and child nodes. The following properties and child nodes are available:

cq:editConfig node properties:

- **cq:actions** (String array): defines the actions that can be performed on the component.
- **cq:layout** (String): : defines how the component is edited.
- **cq:dialogMode** (String): defines how the component dialog is opened.
- **cq:emptyText** (String): defines text that is displayed when no visual content is present.
- **cq:inherit** (Boolean): defines if missing values are inherited from the component that it inherits from.

cq:editConfig child nodes:

- **cq:dropTargets** (node type **nt:unstructured**): defines a list of drop targets that can accept a drop from an asset of the content finder.
- **cq:actionConfigs** (node type **nt:unstructured**): defines a list of new actions that are appended to the cq:actions list.
- **cq:formParameters** (node type **nt:unstructured**): defines additional parameters that are added to the dialog form.
- **cq:inplaceEditing** (node type **cq:InplaceEditingConfig**): defines an inplace editing configuration for the component.
- **cq:listeners** (node type **cq:EditListenersConfig**): defines what happens before or after an action occurs on the component.

NOTE

In this page, a node (properties and child nodes) is represented as XML, as shown in the following example.

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
    cq:actions="[edit]"
    cq:dialogMode="floating"
    cq:layout="editbar"
    jcr:primaryType="cq:EditConfig">
    <cq:listeners
        jcr:primaryType="cq:EditListenersConfig"
        afteredit="REFRESH_PAGE"/>
</jcr:root>
```

There are many existing configurations in the repository. You can easily search for specific properties or child nodes:

- To look for a property of the **cq:editConfig** node, e.g. **cq:actions**, you can use the Query tool in **CRXDE Lite** and search with the following XPath query string:
  //element(cq:editConfig, cq:EditConfig)[@cq:actions]
- To look for a child node of **cq:editConfig**, e.g. **cq:dropTargets** which is of type **cq:DropTargetConfig**, you can use the Query tool in **CRXDE Lite** and search with the following XPath query string:
  //element(cq:dropTargets, cq:DropTargetConfig)

Configuring with cq:EditConfig Properties

CQ:ACTIONS

The **cq:actions** property (String array) defines one or several actions that can be performed on the component. The following values are available:

Property Value	Description
text:<some text>	Displays the static text value <some text>
-	Adds a spacer

edit	Adds a button to edit the component
delete	Adds a button to delete the component
insert	Adds a button to insert a new component before the current one
copymove	Adds a button to copy and cut the component

The following configuration adds an edit button, a spacer, a delete and an insert button to the component edit bar:

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
    cq:actions="[edit,-,delete,insert]"
    cq:layout="editbar"
    jcr:primaryType="cq:EditConfig"/>
```

The following configuration adds the text "Inherited Configurations from Base Framework" to the component edit bar:

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
    cq:actions="[text:Inherited Configurations from Base Framework]"
    cq:layout="editbar"
    jcr:primaryType="cq:EditConfig"/>
```

CQ:LAYOUT

The **cq:layout** property (String) defines how the component can be edited. The following values are available:

Property Value	**Description**
rollover	(default value). The component edition is accessible "on mouse over" through clicks and/or context menu. For advanced use, note that the corresponding client side object is: CQ.wcm.EditRollover.
editbar	The component edition is accessible through a toolbar. For advanced use, note that the corresponding client side object is: CQ.wcm.EditBar.
auto	The choice is left to the client side code.

The following configuration adds an edit button to the component edit bar:

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
    cq:actions="[edit]"
    cq:layout="editbar"
    jcr:primaryType="cq:EditConfig">
</jcr:root>
```

CQ:DIALOGMODE

The component can be linked to an edit dialog. The **cq:dialogMode** property (String) defines how the component dialog will be opened. The following values are available:

Property Value	Description
floating	The dialog is floating.
inline	(default value). The dialog is anchored over the component.
auto	If the component width is smaller than the client side CQ.themes.wcm.EditBase.INLINE_MINIMUM_WIDTH value, the dialog is floating, otherwise it is inline.

The following configuration defines an edit bar with an edit button, and a floating dialog:

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
    cq:actions="[edit]"
    cq:dialogMode="floating"
    cq:layout="editbar"
    jcr:primaryType="cq:EditConfig">
</jcr:root>
```

CQ:EMPTYTEXT

The **cq:emptyText** property (String) defines text that is displayed when no visual content is present.
It defaults to: "Drag components or assets here".

CQ:INHERIT

The **cq:inherit** property (boolean) defines if missing values are inherited from the component that it inherits from.
It defaults to false.

Configuring with cq:EditConfig Child Nodes

CQ:DROPTARGETS

The **cq:dropTargets** node (node type **nt:unstructured**) defines a list of drop targets that can accept a drop from an asset of the content finder. It serves as a collection of nodes of type **cq:DropTargetConfig**.

Each child node of type **cq:DropTargetConfig** defines a drop target in the component. The node name is important because it must be used in the JSP as follows to generate the CSS class name assigned to the DOM element that is the effective drop target:

<drop target css class> = <drag and drop prefix> + <node name of the drop target in the edit configuration>
The <drag and drop prefix> is defined by the Java property com.day.cq.wcm.api.components.DropTarget.CSS_CLASS_PREFIX.
For example, the class name is defined as follows in the JSP of the Download component (/libs/foundation/components/download/download.jsp):
    String ddClassName = DropTarget.CSS_CLASS_PREFIX + "file";
"file" being the node name of the drop target in the edit configuration of the Download component.

The node of type **cq:DropTargetConfig** needs to have the following properties:

Property Name	Property Value

accept	Regex applied to the asset mime type to validate if dropping is allowed.
groups	Array of drop target groups. Each group must match the group type that is defined in the content finder extension and that is attached to the assets.
propertyName	Name of the property that will be updated after a valid drop.

The following configuration is taken from the Download component. It enables any asset (the mime-type can be any string) from the media group to be dropped from the content finder into the component. After the drop, the component property fileReference is being updated:

```
    <cq:dropTargets jcr:primaryType="nt:unstructured">
        <file
            jcr:primaryType="cq:DropTargetConfig"
            accept="[.*]"
            groups="[media]"
            propertyName="./fileReference"/>
    </cq:dropTargets>
```

CQ:ACTIONCONFIGS

The **cq:actionConfigs** node (node type **nt:unstructured**) defines a list of new actions that are appended to the list defined by the **cq:actions** property. Each child node of **cq:actionConfigs** defines a new action by defining a widget. The default widget type is **CQ.Ext.Button**.

The following sample configuration defines two new buttons:

- a separator, defined by the xtype tbseparator.
- a button named "**Manage comments**" that runs the function CQ_collab_forum_openCollabAdmin().

```
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
 xmlns:nt="http://www.jcp.org/jcr/nt/1.0"
    cq:actions="[EDIT,COPYMOVE,DELETE,INSERT]"
    jcr:primaryType="cq:EditConfig">
    <cq:actionConfigs jcr:primaryType="nt:unstructured">
        <separator0
            jcr:primaryType="nt:unstructured"
            xtype="tbseparator"/>
        <manage
            jcr:primaryType="nt:unstructured"
            handler="function(){CQ_collab_forum_openCollabAdmin();}"
            text="Manage comments"/>
    </cq:actionConfigs>
</jcr:root>
```

CQ:FORMPARAMETERS

The **cq:formParameters** node (node type **nt:unstructured**) defines additional parameters that are added to the dialog form. Each property is mapped to a form parameter.

The following configuration adds a parameter called name, set with the value photos/primary to the dialog form:

```
    <cq:formParameters
        jcr:primaryType="nt:unstructured"
        name="photos/primary"/>
```

CQ:INPLACEEDITING

The **cq:inplaceEditing** node (node type **cq:InplaceEditingConfig**) defines an inplace editing configuration for the component. It can have the following properties:

Property Name	Property Value
active	(boolean) True to enable the inplace editing of the component.
configPath	(String) Path of the editor configuration. The configuration can be specified by a configuration node.
editorType	(String) Editor type. The available types are: • plaintext: to be used for non HTML content. • title: is an enhanced plaintext editor that converts graphical titles into a plaintext before editing begins. Used by the Geometrixx title component. • text: to be used for HTML content (uses the Rich Text Editor).

The following configuration enables the inplace editing of the component and defines plaintext as the editor type:

```
<cq:inplaceEditing
    jcr:primaryType="cq:InplaceEditingConfig"
    active="{Boolean}true"
    editorType="plaintext"/>
```

CQ:LISTENERS

The **cq:listeners** node (node type **cq:EditListenersConfig**) defines what happens before or after an action on the component. It can have the following properties:

Property Name	Property Value	Default Value
beforedelete	The handler is triggered before the component is removed.	
beforeedit	The handler is triggered before the component is edited.	
beforecopy	The handler is triggered before the component is copied.	
beforeinsert	The handler is triggered before the component is inserted.	
beforemove	The handler is triggered before the component is moved.	
beforechildinsert	The handler is triggered before the component is inserted inside another component (containers only).	
afterdelete	The handler is triggered after the component is removed.	REFRESH_SELF
afteredit	The handler is triggered after the component is edited.	REFRESH_SELF

aftercopy	The handler is triggered after the component is copied.	REFRESH_SELF
afterinsert	The handler is triggered after the component is inserted.	REFRESH_INSERTED
aftermove	The handler is triggered after the component is moved.	REFRESH_SELFMOVED
afterchildinsert	The handler is triggered after the component is inserted inside another component (containers only).	

NOTE

In the case of nested components there are certain restrictions on actions defined as properties on the cq:listeners node:

- For nested components, the values of the following properties *must* be REFRESH_PAGE:
  - aftermove
  - aftercopy

The event handler can be implemented with a custom implementation. For example:
afteredit = "project.customerAction"
where project.customerAction is a static method.

The following example is equivalent to the REFRESH_INSERTED configuration:
afterinsert="function(path, definition) { this.refreshCreated(path, definition); }"

To know the parameters that can be used in the handlers, refer to the before<action> and after<action> events section of the CQ.wcm.EditBar and CQ.wcm.EditRollover widget documentation.

With the following configuration the page is refreshed after the component has been deleted, edited, inserted or moved:

```
<cq:listeners
    jcr:primaryType="cq:EditListenersConfig"
    afterdelete="REFRESH_PAGE"
    afteredit="REFRESH_PAGE"
    afterinsert="REFRESH_PAGE"
    afterMove="REFRESH_PAGE"/>
```

A closer look at a few of the foundation components...

The following sections explain how the most commonly used components of the reference website Geometrixx have been developed.

TOP NAVIGATION COMPONENT

The Top Navigation Component displays the top level pages of the website. This navigation component is placed at the top of the content pages in the website.
There is no content to be handled by this component. Therefore there is no need for a dialog, only rendering.
Specification summary:

- /libs/foundation/components/topnav
- Displays level one pages (below Homepage).
- Respects On/Off status and uses image rendering.
- Displays as in the following screenshot:

Navigation Essentials

The main function of a navigation component is to:

- show the hierarchical page structure of a website
- provide the possibility to access the pages within this structure.

To achieve this, functionality to get the childpages of a specific page is essential. Also required is the functionality to: get the path of the parent page of any page on a specific absolute level (start page of the navigation), check the validity of a page and of course get the path and title of a page.
The following are an introduction to the relevant functionality (API calls):

- com.day.cq.wcm.core.PageManager.getPage(String path): get the page related to a path
- com.day.cq.wcm.core.Page.listChildren(): get the childpages of the page
- com.day.cq.wcm.core.Page.getPath() + .getTitle(): get the path or title of the page respectively
- com.day.cq.wcm.core.Page.isValid() + .isHideInNav(): checks if the page is valid or the hide in navigation property is set respectively
- com.day.cq.wcm.core.Page.getAbsoluteParent(int level): Get the parent page on an absolute level
- PageFilter(): The default AEM Page filter: checks if the page is valid (by checking that the property hideInNav is not set, performing checks on On-/Offtime etc); can be used instead of isValid() and isHideInNav()

For more detailed information please have a look at the Javadoc provided with AEM WCM.

Image Rendering Essentials

To be able to use image based navigation items, a mechanism is needed to request a page in an image navigation item view.
To achieve this a specific selector is added to the request for the navigation item image of a page; for example, /path/to/a/page.navimage.png. Requests with such a selector have to be handled by an image processing mechanism. Sling's request processing mechanism is used for this. To realize this, an image processing script (or servlet) is added, this handles all requests with the specific selector (for example, /contentpage/navimage.png.jsp or /contentpage/navimage.png.java).
For rendering text images the abstract servlet AbstractImageServlet is very helpful. By overwriting the createLayer() method, it is possible to programmatically create a fully customized image.
The following lists the relevant functionalities (API calls):

- com.day.cq.wcm.commons.AbstractImageServlet
- com.day.cq.wcm.commons.WCMUtils
- com.day.cq.wcm.foundation.ImageHelper
- com.day.image.Font
- com.day.image.Layer

For more detailed information please have a look at the Javadoc provided with AEM WCM.

Image based Top Navigation Component

The Top Navigation component renders graphical (image based) navigation items.
The images for the navigation items are requested from the page resources in a specific view. This means, the paths of the image tags for the navigation items are equivalent to the paths of the pages that the navigation items represent. To identify the view (kind of presentation) the resource (page) is rendered by, a specific URL selector ('navimage') is added.

LIST CHILDREN COMPONENT

The List Children component displays the child pages under a given root page. The root page can be configured for every instance of this component (for every paragraph which is rendered by this component). Therefore a dialog is needed to store the path of the root page as content in the corresponding paragraph resource. If no root page is set, the current page is taken as the root page. The component displays a list of links with title, description and date.
Specification summary:

- Display a list of links with title, description and date, referring to pages which are below either the current page or a root page defined by the path provided in a dialog

- /libs/foundation/components/listchildren
- Respects the On/Off status of displayed pages
- The following screenshot shows an example of how it displays:

Dialog & Widgets

The Dialog of a component is defined in a subtree of nodes below the component's root node. The root node of the dialog is of nodeType cq:Dialog and named dialog. Below this, root nodes for the individual tabs of the dialog are added. These tab nodes are of nodeType cq:WidgetCollection. Below the tab-nodes, the widget nodes are added; these are of nodeType cq:Widget.
Summary:

Location: /apps/<myapp>/<mycomponent>/dialog
Root Node:
- dialog (cq:Dialog) - node for the dialog

Vital Properties:
- xtype=panel - Defines the dialog's xtype as panel; title sets the title of the dialog

Vital Child Nodes of Dialog Node:
- items (cq:WidgetCollection) - tab nodes within the dialog

Vital Child Nodes of Tab Node:
- <mywidget> (cq:Widget) - widget nodes within the tab

Vital Properties:
- name - Defines the name of the property where the content provided by this widget is stored (usually something like ./mypropertyname)
- xtype - Defines the widget's xtype
- fieldLabel - The text displayed in the dialog as a label for this widget

LOGO COMPONENT

The Logo component displays the logo of the website Geometrixx. The logo image and the home link can be configured globally (same for every page of the website) so that every instance of this component is identical. Therefore a design dialog is needed to provide the image and path of the home link to the design of the corresponding Page. The Logo component is placed in the upper left corner of all pages on the website.
Specification summary:
- /libs/foundation/components/logo
- Displays a linked logo image in the upper left corner (spooled image, no rendering)
- The path for the link is the path of a page on a defined absolute level
- The logo image and the level are the same for all pages on the website; store the logo image and level in the design of geometrixx
- Displays as in the following screenshot

Designer

The Designer is used to manage the look-and-feel of the global content; including the path to the tool-pages, image of the logo, design values such as text family, size and so on.
Summary:

Location: /etc/designs
Root Node:
- <mydesign> (cq:Page) - Hierarchy node of the design page

Vital Child Nodes:

- jcr:content (cq:PageContent) - Content node for the design

Vital Properties of Child Node jcr:content:

- sling:resourceType = “wcm/designer” - Reference to the designer rendering component

The Designer values can be accessed by the currentStyle object provided in global.jsp.
Design dialogs are structured in the same way as normal dialogs but are named design_dialog.

PARAGRAPH SYSTEM

The Paragraph System is a key part of a website as it manages a list of paragraphs. It is used to structure the individual pieces of content on a website. You can create paragraphs inside the Paragraph System, move, copy and delete paragraphs and also use a column control to structure your content in columns. The Paragraph System provided in AEM WCM foundations covers most of the variants needed and can also be configured by allowing you to select the components to be activated/deactivated within your current paragraph system.

IMAGE COMPONENT

The Image component displays images in the main paragraph system.
Specification summary:

- /libs/foundation/components/image
- Displays an image in the main paragraph system.
- The image and certain paragraph-related display properties (title, description, size) are stored in the paragraph resource by a dialog.
- Some global design properties, valid for all paragraphs of this type (minimal size, maximal size), are stored in the design by a design dialog.
- There is the possibility to crop, map etc the image.

Widget smartimage

The smartimage widget is an extended widget used to handle the most common aspects of image handling in a WCM system. It controls the upload of the image file and stores the reference to the media library. It also supports the cropping and mapping of images amongst other functions.
The most important properties of the smartimage widget are:

- xtype - The type of widget ('smartimage').
- name - The place to store the image file (binary), usually ./image/file or ./file.
- title - The title displayed in the dialog.
- cropParameter - The place to store the crop coordinates, usually ./image/imageCrop or ./imageCrop.
- ddGroups - Groups in contentfinder from where assets can be dragged to this widget.
- fileNameParameter - Specifies where the name of the image file will be stored, usually ./image/fileName or ./fileName.
- fileReferenceParameter - Location to store the image reference when an image from the media library is used, usually ./image/fileReference or ./fileReference.
- mapParameter - Where to store the map data, usually ./image/imageMap or ./imageMap.
- requestSuffix - The default suffix to be used when browsing for an image with this widget, usually ./image.img.png or ./img.png.
- rotateParameter - Where to store the rotation specification, usually ./image/imageRotate or ./imageRotate.
- sizeLimit - Maximum size limit, i.e. 100.
- uploadUrl - The path to be used when storing data temporarily, usually /tmp/uploaded_test/*.

Contentfinder Essentials

To be able to drag assets from the contentfinder to a component there must be a drop target configuration node called cq_dropTargets below the edit configuration node (cq:editConfig).
Summary:
Location: /apps/<myapp>/components/<mycomponent>/cq:editConfig/cq:dropTargets
Root node:

- cq:dropTargets (cq:DropTargetConfig) - Hierarchy Node of the drop target configuration.

Below this node the following node tree (with vital properties) must be created for the smartimage:
- image (nt:unstructured) with the following vital properties:
  - accept - The media types to be accepted; i.e. image/gif, image/jpeg or image/png.
  - groups - Groups in the contentfinder from where assets can be accepted, i.e. media
  - propertyName - Where the reference will be stored; usually ./image/fileReference or ./fileReference

In the case that the image is stored in a separate image node in the content (for example, as with textimage, when the image is not the only data to be stored for the resource) the following two nodes must also be created:
- parameters (nt:unstructured) below the image node
- image (nt:unstructured) below the parameters node with the following vital properties:
  - sling:resourceType - The resource type to be stored in the case that a complete paragraph has to be created (as when dragging an asset to the paragraphsystem while pressing the Alt button).

Image Component Essentials

This section lists the most relevant functionalities (API calls) for the programmatic manipulation of images:
- com.day.cq.wcm.foundation.Image
  - addCssClass
  - loadStyleData
  - setSelector
  - setSuffix
  - draw
  - getDescription
- com.day.cq.wcm.api.components.DropTarget
- com.day.cq.wcm.api.components.EditConfig
- com.day.cq.wcm.commons.WCMUtils

For more detailed information please look at the Javadoc provided with AEM WCM.

Image Rendering Essentials

As for the Top Navigation component, the AbstractImageServlet is used to render the images. For an introduction to the AbstractImageServlet, see the Image Rendering Essentials in the Top Navigation Component section.
As the request to the resource in the 'image view' has to be detected, a specific selector to the request of the image (i.e. /path/to/the/resource.img.png) needs to be added. Requests with such a selector are handled by the image processing servlet.

TEXT IMAGE COMPONENT

The Text Image Component displays text and images in the main paragraph system.
Specification summary:
- /libs/foundation/components/textimage
- Displays a text and image in the main paragraph system.
- The text and image together with specific paragraph related display properties (title, description, size) are stored in the paragraph resource (dialog).
- There is the possibility to manipulate the image, including cropping and mapping amongst other functions.
- For the image rendering use the servlet created for the image component (to inherit from the image component set resourceSuperType)
- Text input uses the Rich Text Editor, which can be fully configured.
- On the **Advanced Image Properties** tab the **Style** can be configured to list styles defined in your CSS file. Often these are used to left or right align the image.

Widget richtext

The richtext widget is an extended widget used to handle the most common aspects of text handling in a WCM system. It stores the text with the formatting information.
The most important properties of the richtext widget are:

- xtype - The type of the widget ('richtext')
- name - Where the text is stored, usually ./text.
- hideLabel - Defines whether the label should be displayed, usually false.
- richFlag (xtype=hidden) with name ./textIsRich, ignoreData=true and value=true - Used to define that format is rich text format.

Text & Image Component Essentials

This section lists the most relevant functionalities (API calls) for the programmatic manipulation of texts and images:

- com.day.cq.wcm.foundation.TextFormat
- com.day.cq.wcm.api.WCMMode

For more detailed information please have a look at the Javadoc provided with AEM WCM.

SEARCH COMPONENT

The Search component can be placed in the paragraph system of any page. It searches the content of the site for a query provided in the request.
Specification summary:

- /libs/foundation/components/search
- Displays a search form.
- Displays the result of the search for a query provided in the request (if a query was provided).
- Provides a dialog to define some properties
  - Search button text
  - No results text
  - Previous label
  - Next label
- Provides pagination

Search Essentials

This section lists the most relevant functionalities (API calls) for the programmatic manipulation of searches:

- com.day.cq.wcm.foundation.Search - Search Class to be used for almost every aspect of the search. The query is expected in a request parameter named 'q'
  - getResult - Get the result object
- com.day.cq.wcm.foundation.Search.Result
  - getResultPages
  - getPreviousPage
  - getNextPage

For more detailed information please have a look at the API documentation provided with AEM WCM.