CQ5 WCM Developer's Guide

CQ5 WCM Developer's Guide

Day

Contents

1 Introduction

1.1 Introduction

Day's CQ5 platform allows you to build compelling content-centric applications that combine Web Content Management, Workflow Management, Digital Asset Management and Social Collaboration.

The product has been completely redesigned from Communiqué 4, allowing Day to use new architecture and technologies, thus increasing functionality while reducing complexity. Extensive use of standards helps ensure long-term stability.

1.2 Purpose of this Document

To provide the information necessary for developing new components, applications and other elements within CQ.

Warning

This document is not a programming guide; it is assumed that you have the necessary skills for writing code in the appropriate language.

Important

This document is designed to be read in conjunction with the CQ5 WCM Architect Guide.

1.3 Target Audience

- Developers

1.4 Prerequisites for development within CQ

For development within CQ, you need the following skills:

- Basic knowledge of web application techniques, including:
    - the request -response (XMLHttpRequest / XMLHttpResponse) cycle
    - HTML
    - CSS
    - JavaScript
- Working knowledge of CRX; including the Content Explorer
- Basic knowledge of JSP (JavaServer Pages), with the ability to understand and modify simple JSP examples

2 CQ in-depth

2.1 JSR-170 and the JCR API

JSR 170 is the Java Specification Request for the Content Repository for JavaTM technology API. Specification lead is held by Day Software AG.

The JCR API package, `javax.jcr.*` is used for the direct access and manipulation of repository content.

CRX is Day's proprietary implementation of the JCR.

Apache Jackrabbit is an open source, fully conforming, implementation of this API.

2.2 CQ DAM

CQ DAM (Communiqué Digital Asset Management) is used to centrally manage all digital media files and essential metadata information.

2.3 Widgets

CQ WCM has been developed using the ExtJS library of widgets.

2.4 FileVault (source revision system)

FileVault provides your JCR repository with file system mapping and version control. It can be used to manage CQ development projects with full support for storing and versioning project code, content, configurations and so on, in standard version control systems (for example, Subversion).

2.5 Workflow Engine

Your content is often subject to organizational processes, including steps such as approval and sign-off by various participants. These processes can be represented as workflows, defined within CQ, then applied to the appropriate content pages or digital assets as required.

The Workflow Engine is used to manage the implementation of your workflows, and their subsequent application to your content. More information on how to use Workflows is given in the Chapter 7, .

2.6 Dispatcher

The Dispatcher is Day's tool for both caching and/or load balancing. Further information can be found under Tools - the Dispatcher.

2.7 Localization

Localization is at the core of CQ5. It provides support for adapting applications, created using the CQ5 platform, into different languages and regional configurations . While processing the request, the *Locale* is extracted. This is then used to reference a language code, and optionally a country code, which can be used for controlling either the specific content or format of certain output.

Localization will be used throughout CQ5 - wherever reasonable. One notable exception is the system log information of CQ5 itself, this is never localized and always in English.

2.8 Sling Request Processing

2.8.1 Introduction to Sling

CQ5 is built using Sling, a Web application framework based on REST principles that provides easy development of content-oriented applications. Sling uses a JCR repository, such as Apache Jackrabbit, or Day's CRX, as its data store.

Sling started as an internal project of Day Management AG and is included in the installation of CQ5. Sling has since been contributed to the Apache Software Foundation - further information can be found at Apache.

Using Sling, the type of content to be rendered is not the first processing consideration. Instead the main consideration is whether the URL resolves to a content object for which a script can then be found to perform the rendering. This provides excellent support for web content authors to build pages which are easily customized to their requirements.

The advantages of this flexibility are apparent in applications with a wide range of different content elements, or when you need pages that can be easily customized. In particular, when implementing a Web Content Management system such as CQ WCM.

2.8.2 Sling is Content Centric

Sling is **content-centric**. This means that processing is focused on the content as each (HTTP) request is mapped onto content in the form of a JCR resource (a repository node):

- the first target is the resource (JCR node) holding the content
- secondly, the representation, or script, is located from the resource properties in combination with certain parts of the request (e.g. selectors and/or the extension)

2.8.3 RESTful Sling

Due to the content-centric philosophy, Sling implements a REST-oriented server and thus features a new concept in web application frameworks. The advantages are:

- very RESTful; resources and representations are correctly modelled inside the server
- removes one or more data models
  - previously the following were needed: URL structure, business objects, DB schema;
  - this is now reduced to: URL = resource = JCR structure

2.8.4 URL Decomposition

In Sling, and therefore also CQ5, 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:

```
http://myhost/tools/spy.printable.a4.html/a/b?x=12
```

We can break it down into its composite parts:

Table 2.1. URL Decomposition

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

protocol
HTTP.

host
Name of the website.

content path
Path specifying the content to be rendered. Is used in combination with the extension; in this example they translate to `tools/spy.html`.

selector(s)
Used for alternative methods of rendering the content; in this example a printer-friendly version in A4 format.

extension
Content format; also specifies the script to be used for rendering.

suffix
Can be used to specify additional information.

param(s)
Any parameters required for dynamic content.

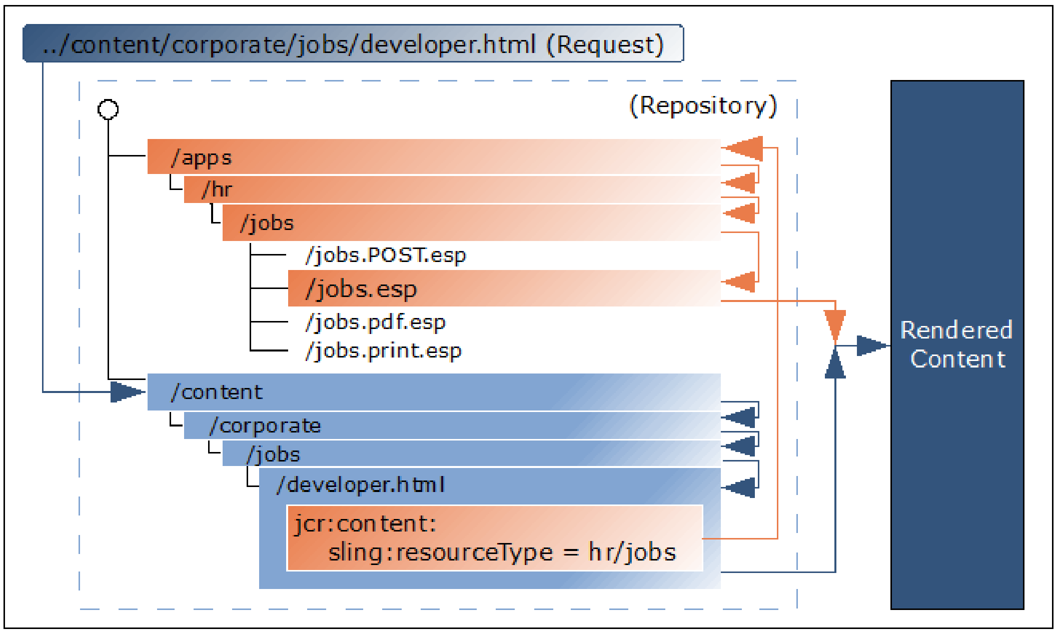
2.8.5 From URL to Content and Scripts

Using these principles:

- the mapping uses the *content path* extracted from the request to locate the resource
- when the appropriate resource is located, the *sling resource type* is extracted, and used to locate the script to be used for rendering the content

The figure below illustrates the mechanism used, which will be discussed in more detail in the following sections.

Figure 2.1. How Sling locates the content and scripts



Therefore:

- DO NOT specify which data entities to access in your scripts (as an SQL statement in a PHP script would do)
- DO specify which script renders a certain entity (by setting the `sling:resourceType` property in the JCR node)

2.8.5.1 Mapping requests to resources

The request is broken down and the necessary information extracted. The repository is searched for the requested resource (content node):

- first Sling checks whether a node exists at the location specified in the request; e.g. `../content/corporate/jobs/developer.html`
- if no node is found, the extension is dropped and the search repeated; e.g. `../content/corporate/jobs/developer`
- if no node is found then Sling will return the http code 404 (Not Found).

Note

Sling also allows things other than JCR nodes to be resources, but this is an advanced feature.

2.8.5.2 Locating the script

When the appropriate resource (content node) is located, the *sling resource type* is extracted. This is a path, which locates the script to be used for rendering the content.

The path specified by the *`sling:resourceType`* can be either:

- absolute
- relative, to a configuration parameter

Note

Relative paths are recommended by Day as they increase portability.

All Sling scripts are stored in subfolders of either `/apps` or `/libs`, which will be searched in this order.

A few other points to note are:

- when the Method (GET, POST) is required, it will be specified in uppercase as according to the HTTP specification e.g. jobs.POST.esp (see below)
- various script engines are supported:
  - `.esp, .ecma`: ECMAScript (JavaScript) Pages (server-side execution)
  - `.jsp`: Java Server Pages (server-side execution)
  - `.java`: Java Servlet Compiler (server-side execution)
  - `.jst`: JavaScript templates (client-side execution)
  - `.js`: ECMAScript / JavaScript (client-side execution)

The list of script engines supported by the given instance of CQ are listed on the Felix Management Console (`http://localhost:4502/system/console/scriptengines`).

Additionally, Apache Sling supports integration with other popular scripting engines (e.g., Groovy, JRuby, Freemarker), and provides a way of integrating new scripting engines.

Using the above example, if the *`sling:resourceType`* is `hr/jobs` then for:

- GET/HEAD requests, and URLs ending in .html (default request types, default format)

  The script will be `/apps/hr/jobs/jobs.esp`; the last section of the *`sling:resourceType`* forms the file name.

- POST requests (all request types excluding GET/HEAD, the method name must be uppercase)

  POST will be used in the script name.

  The script will be `/apps/hr/jobs/POST.esp`.

- URLs in other formats, not ending with .html

  For example `../content/corporate/jobs/developer.pdf`

  The script will be `/apps/hr/jobs/jobs.pdf.esp`; the suffix is added to the script name.

- URLs with selectors

  Selectors can be used to display the same content in an alternative format. For example a printer friendly version, an rss feed or a summary.

  If we look at a printer friendly version where the selector could be **print**; as in `../content/corporate/jobs/developer.print.html`

  The script will be `/apps/hr/jobs/jobs.print.esp`; the selector is added to the script name.

- If no *`sling:resourceType`* has been defined then:

  - the content path will be used to search for an appropriate script (if the path based ResourceTypeProvider is active).

    For example, the script for `../content/corporate/jobs/developer.html` would generate a search in `/apps/content/corporate/jobs/`.

  - the primary node type will be used.

- If no script is found at all then the default script will be used.

  The default rendition is currently supported as plain text (.txt), HTML (.html) and JSON (.json), all of which will list the node's properties (suitably formatted). The default rendition for the extension `.res`, or requests without a request extension, is to spool the resource (where possible).

- For http error handling (codes 404 or 500) Sling will look for a script at `/libs/sling/servlet/errorhandler/404.esp`, or `500.esp`, respectively.

If multiple scripts apply for a given request, the script with the best match is selected. The more specific a match is, the better it is; in other words, the more selector matches the better, regardless of any request extension or method name match.

For example, consider a request to access the resource `/content/corporate/jobs/developer.print.a4.html` of type *`sling:resourceType="hr/jobs"`*. Assuming we have the following list of scripts in the correct location:

1. jobs.esp

2. jobs.GET.esp

3. jobs.GET.html.esp
4. jobs.html.esp
5. jobs.print.esp
6. jobs.print.a4.esp
7. jobs.print.html.esp
8. jobs.print.GET.html.esp
9. jobs.print.a4.html.esp
10. jobs.print.a4.GET.html.esp

Then the order of preference would be (10) - (9) - (6) - (8) - (7) - (5) - (3) - (4) - (2) - (1).

Note

(6) is a better match than (8), because it matches more selectors even though (8) has a method name and extension match where (6) does not.

In addition to the resource types (primarily defined by the *`sling:resourceType`* property) there is also the resource super type. This is generally indicated by the *`sling:resourceSuperType`* property. These super types are also considered when trying to find a script. The advantage of resource super types is that they may form a hierarchy of resources where the default resource type `sling/servlet/default` (used by the default servlets) is effectively the root.

The resource super type of a resource may be defined in two ways:

1. by the *`sling:resourceSuperType`* property of the resource.
2. by the *`sling:resourceSuperType`* property of the node to which the *`sling:resourceType`* points.

   For example:

   - /
     - a
     - b
       - sling:resourceSuperType = a
     - c
       - sling:resourceSuperType = b
     - x
       - sling:resourceType = c
     - y
       - sling:resourceType = c
       - sling:resourceSuperType = a

   The type hierarchy of `/x` is `[ c, b, a, <default> ]` while for `/y` the hierarchy is `[ c, a, <default> ]` because `/y` has the *`slingresourceSuperType`* property whereas `/x` does not and therefore its supertype is taken from its resource type.

2.8.5.2.1 Sling Scripts cannot be called directly

Within Sling, scripts cannot be called directly as this would break the strict concept of a REST server; you would mix resources and representations.

If you call the representation (the script) directly you hide the resource inside your script, so the framework (Sling) no longer knows about it. Thus you lose certain features:

- automatic handling of http methods other than GET, including:
  - POST, PUT, DELETE which are handled with a sling default implementation
  - the POST.js script in your sling:resourceType location
- your code architecture is no longer as clean nor as clearly structured as it should be; of prime importance for large-scale development

2.8.6 Sling API

This uses the Sling API package, `org.apache.sling.*`, and tag libraries.

2.8.7 Referencing existing elements using sling:include

A final consideration is the need to reference existing elements within the scripts.

More complex scripts (aggregating scripts) might need to access multiple resources (for example navigation, sidebar, footer, elements of a list) and do so by *including* the **resource**.

To do this you can use the `sling:include("/<path>/<resource>")` command. This will effectively include the definition of the referenced resource, as in the following statement which references an existing definition for rendering images:

```
%><sling:include resourceType="geometrixx/components/image/img"/><%
```

2.8.8 First Steps - an example for using Sling

An introduction the first steps of developing with Sling (and CRX) can be seen on `http://dev.day.com/`.

2.9 OSGI

OSGi defines an architecture for developing and deploying modular applications and libraries (it is also known as the Dynamic Module System for Java). OSGi containers allow you to break your application into individual modules (are jar files with additional meta information and called *bundles* in OSGi terminology) and manage the cross-dependencies between them with:

- *services* implemented within the container
- a *contract* between the container and your application

These services and contracts provide an architecture which enables individual elements to dynamically discover each other for collaboration.

An OSGi framework then offers you dynamic loading/unloading, configuration and control of these bundles - without requiring restarts.

Note

Full information on OSGi technology can be found at the OSGi Alliance Technology Overview.

In particular, their Basic Education page holds a collection of presentations and tutorials.

This architecture allows you to extend Sling with application specific modules. Sling, and therefore CQ5, uses the Apache Felix implementation of OSGI (Open Services Gateway initiative). They are both collections of OSGi bundles running within an OSGi framework.

This enables you to perform the following actions on any of the packages within your installation:

- install
- start
- stop
- update
- uninstall
- see the current status
- access more detailed information (e.g. symbolic name, version, location, etc) about the specific bundles

3 CQ5 WCM - Architecture and Concepts

CQ5 is Day's suite of products based on a standard JCR repository. WCM is the Web Content Management solution within CQ.

This section should be read in conjunction with the CQ5 WCM Architect Guide, which gives an overview of the architecture and related concepts within CQ5.

3.1 Development objects

The following are of interest at the development level:

Node (and their properties)

Nodes and their properties store content, CQ object definitions, rendering scripts and other data.

Nodes define the content structure, and their properties store the actual content and metadata.

Content nodes drive the rendering. Sling gets the content node from the incoming request. The property sling:resourceType of this node points to the Sling rendering component to be used.

A node, which is a JCR name, is also called a resource in the Sling environment.

Note

In CQ, everything is stored in the repository.

Widget

In CQ all user input is managed by widgets. These are often used to control the editing of a piece of content.

Dialogs are built by combining Widgets.

Dialog

A dialog is a special type of widget.

To edit content, CQ uses dialogs defined by the application developer. These combine a series of widgets to present the user with all fields and actions necessary to edit the related content.

Dialogs are also used for editing metadata, and by various administrative tools.

Component

A software component is a system element offering a predefined service or event, and able to communicate with other components.

Within CQ a component is often used to render the content of a resource. When the resource is a page, the component rendering it is called a Top-Level Component or a Pagecomponent. However, a component does not have to render content, nor be linked to a specific resource; for example, a navigation component will display information about multiple resources.

The definition of a component includes:,

- the code used to render the content
- a dialog for the user input and the configuration of the resulting content.

Template

A template is the blueprint for a specific type of page. When creating a page in the siteadmin the user has to select a template. The new page is then created by copying this template.

A template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content.

It defines the page component used to render the page and the default content (primary top-level content). The content defines how it is rendered as CQ is content-centric.

Page Component (Top-Level Component)
: The component to be used to render the page.

Page
: A page is an 'instance' of a template.

A page has a hierarchy node of type cq:Page and a content node of type cq:PageContent. The property sling:resourceType of the content node points to the Page Component used for rendering the page.

3.2 Structure within the repository

The following list gives an overview of the structure you will see within the repository.

Warning

Changes to this structure, or the files within it, should be made with care.

Changes are needed when you are developing, but you should take care that you fully understand the implications of any changes you make.

Warning

You must not change anything in the `libs/` path. For configuration and other changes copy the item in `libs/` to `apps/` and make any changes within `apps/`.

- `/apps`

  Application related; includes component definitions specific to your website.

- `/content`

  Content created for your website.

- `/etc`

  Initialization and configuration information.

- `/home`

  User and Group information.

- `/libs`

  Libraries and definitions that belong to the core of CQ WCM.

- `/tmp`

  Temporary working area.

- `/var`

  Files that change and updated by the system; such as audit logs, statistics, event-handling.

4 Development Tools

4.1 Working with the CQ Development Environment (CQDE)

The CQ5 IDE (CQDE) provides a development platform for CQ5 applications. It is custom-built specifically for CQ and therefore recommended by Day.

CQDE is built on Eclipse RCP and EFS and comes as a set of Eclipse plugins.

4.1.1 Setting up CQDE

To set up CQDE, proceed as follows:

1. Ensure that CQ is installed and running.
2. Download the CQDE package.
3. Extract the package to your required installation location.
4. **`Double-click`** the executable.
5. Enter the location of your CQ5 installation; for example `http://localhost:4502` for CQ5 Quickstart.
6. Enter your **`username`** and **`password`**.
7. Click **`OK`**.

Note

If the location of your installation is different from above (for example, another host, port or context path) you also need to update the CRX server endpoint. This is the location where CQDE can find the CRX WebDAV server. This can be done in the configuration section of the CQDE servlet which is available on the Felix Management Console (`http://localhost:4502/system/console/configMgr`).

To add the /etc folder into the CQDE Navigator panel, proceed as follows:

1. In CRX Explorer: for the node `/libs/cqde/profiles/default`, add **`/etc`** to the property **`mountPath`**.
2. Restart CQDE.

4.1.2 Configuring CQDE

If the location of your installation is different from above (for example, another host, port or context path) you also need to update the CRX server endpoint. This is the location where CQDE can find the CRX WebDAV server. This can be done in the configuration section of the CQDE servlet which is available on the Felix Management Console (`http://localhost:4502/system/console/configMgr`).

When CQDE is started, it sends information about the users environment to CQ5; including version, OS and a profile ID. CQ5 stores this information under `/libs/cqde/profiles`, where each profile must have its own folder. The standard profile is `default`. The profile folder contains XSLT templates for generating CQDE project and classpath files, as well as additional settings.

```
- libs
  - cqde
    - profiles
      - [Name of Profile] (the default profiles that is distributed with CQ5)
        - project.xml.xslt (mandatory template for generating .project files)
        - classpath.xml.xslt (mandatory template for generating .classpath files)
```

```
       + mountPaths (optional multivalue string property defining the paths that should be
mounted, if omitted root is mounted)
       + cqdeVersions (optional multivalue string property defining the CQDE5 versions
that can use this profile)
       + cqdeOS (optional multivalue string property defining the operating systems that
can use this profile)
```

4.2 How to Set Up the Development Environment with Eclipse

This document describes the process of setting up a local development environment for a simple CQ5 project with Eclipse. It then describes how to integrate logic into the project through Java coding and JSP scripting. Lastly, it points to open source software to enable collaborative and automated developing.

The setup described here is an alternative among others and may vary from project to project.

The local development environment involves:

- A CQ5 installation that will act as your local environment.
- CRX Explorer within the CQ5 instance to create and edit nodes and properties within the CRX repository.
- FileVault (VLT), a Day developed utility that maps the CRX repository to your file system.
- Eclipse to edit the project source on your local file system.
- Apache Maven to run local snapshot builds.

4.2.1 Creating the Project Structure in CQ5

This section describes the creation of a simple project structure in CQ5:

1. Install CQ5 on your machine. Please refer to the section called “Installing a CQ WCM instance - Generic procedure” for the detailed procedure. In the current context, CQ5 runs locally on port 4502.

   If already installed then ensure it is running and connect.

2. In the CRX Explorer, create the project structure:

   1. Under the `/apps` folder, create the nt:folder `myApp`.
   2. Under the `myApp` folder, create the nt:folder `components`.
   3. Under the `myApp` folder, create the nt:folder `templates`.
   4. Under the `myApp` folder, create the nt:folder `install`.

3. In your browser, navigate to the **`Miscellaneous`** tab. Under **`designs`**, create the design page of your application:

   - **`Title`**: **`My Application Design Page`**.
   - **`Name`**: **`myApp`**.
   - **`Template`**: **`Design Page Template`**.

4.2.2 Installing FileVault (VLT)

FileVault (VLT) is a tool developed by Day that maps the content of a CRX instance to your file system. The VLT tool has similar functionalities to those of an SVN client, providing normal check in, check out and management operations, as well as configuration options for flexible representation of the project content.

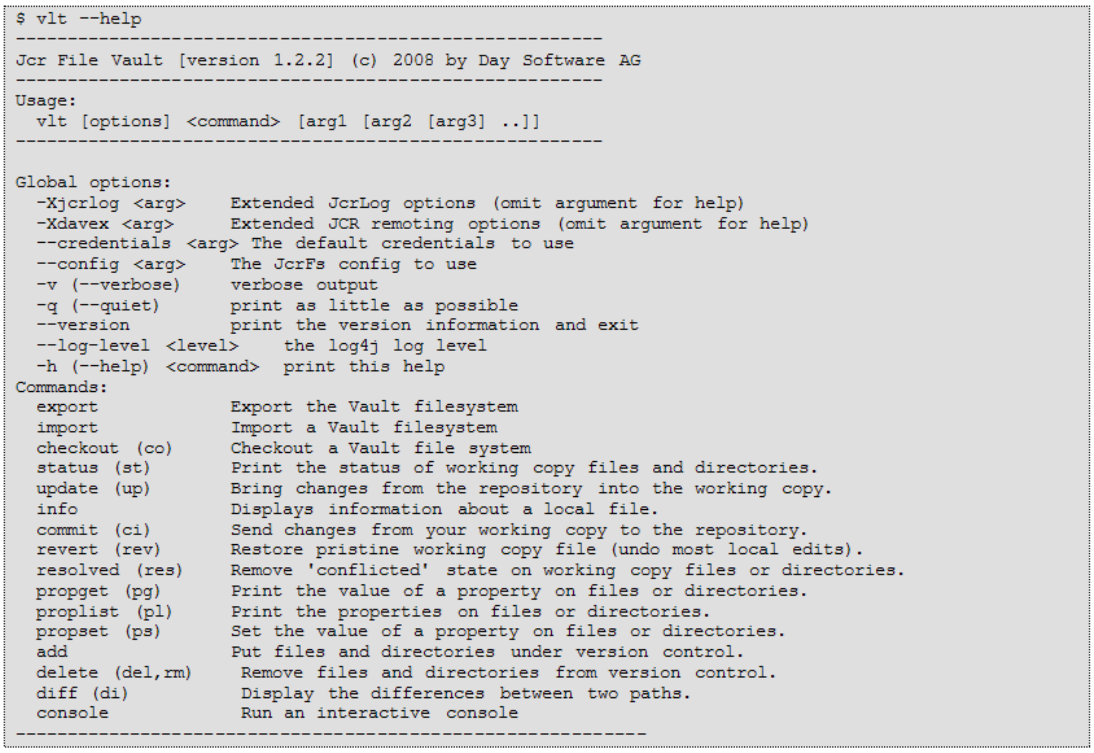
To install VLT, follow the steps:

1. In your file system, go to `<cq-installation-dir>/crx-quickstart/opt/filevault`. The build is available in both tgz and zip formats.

2. Extract the archive.

3. Add `<cq-installation-dir>/crx-quickstart/opt/filevault/vault-cli-<version>/bin` to your environment PATH so that the command files `vlt` or `vlt.bat` are accessed as appropriate. For example, `<cq-installation-dir>/crx-quickstart/opt/filevault/vault-cli-1.1.2/bin`

4. Open a command line shell and execute **vlt --help**. Make sure it displays the following help screen:



```
$ vlt --help
------------------------------------------------------------
Jcr File Vault [version 1.2.2] (c) 2008 by Day Software AG
------------------------------------------------------------
Usage:
  vlt [options] <command> [arg1 [arg2 [arg3] ..]]
------------------------------------------------------------

Global options:
  -Xjcrlog <arg>    Extended JcrLog options (omit argument for help)
  -Xdavex <arg>     Extended JCR remoting options (omit argument for help)
  --credentials <arg> The default credentials to use
  --config <arg>    The JcrFs config to use
  -v (--verbose)    verbose output
  -q (--quiet)      print as little as possible
  --version         print the version information and exit
  --log-level <level>    the log4j log level
  -h (--help) <command>  print this help
Commands:
  export            Export the Vault filesystem
  import            Import a Vault filesystem
  checkout (co)     Checkout a Vault file system
  status (st)       Print the status of working copy files and directories.
  update (up)       Bring changes from the repository into the working copy.
  info              Displays information about a local file.
  commit (ci)       Send changes from your working copy to the repository.
  revert (rev)      Restore pristine working copy file (undo most local edits).
  resolved (res)    Remove 'conflicted' state on working copy files or directories.
  propget (pg)      Print the value of a property on files or directories.
  proplist (pl)     Print the properties on files or directories.
  propset (ps)      Set the value of a property on files or directories.
  add               Put files and directories under version control.
  delete (del,rm)    Remove files and directories from version control.
  diff (di)          Display the differences between two paths.
  console            Run an interactive console
--------------------------------------------------------------
```

4.2.3 Installing Eclipse

Eclipse is open source software used to edit the project source locally on your file system. Apache Maven is also open source software, used to run local snapshot builds: it compiles Java code and stores the compiled code in a jar file.

In this section, you will install Eclipse and a Maven plugin which embeds the Maven functionality within Eclipse:

1. Download Eclipse - select the **`Eclipse IDE for Java EE Developers`** option.

2. Install Eclipse: extract from the downloaded zip file to your destination directory.

3. Start Eclipse:

   a. Navigate to the directory into which you extracted the contents of the Eclipse installation zip file. For example `C:\Program Files\Eclipse\`.

   b. Double-click on `eclipse.exe` (or `eclipse.app`) to start Eclipse.

4. Create a new **`workspace`** for your project and name it **`myApp`**.

5. Install the Maven plugin (m2) from Sonatype. Disable Maven SCM handler for Subclipse (Optional) and Maven Integration for AJDT (Optional).

6. After installation it is recommended to restart Eclipse.

4.2.4 Creating the Project Structure in Eclipse

In this section, you will create 2 Maven projects:

- one called UI (after User Interface) which contains the CQ5 project structure with the JSP scripts.
- the other called Core which contains the Java code (source and compiled). The compiled code is stored in a jar file.

The advantage of such a structure is that it adds modularity and autonomy to the logic your application because each jar file (bundle) can be managed separately.

4.2.4.1 Create the UI Maven Project

To create the UI Maven project, follow the steps:

1. In Eclipse open the Workbench.
2. Create the UI Maven project:
    1. In the Menu bar, click **`File`**, select **`New`**, then **`Other...`** .
    2. In the dialog, expand the **`Maven`** folder, select **`Maven Project`** and click **`Next`**.
    3. Check the **`Create a simple project`** box and the **`Use default Workspace locations`** box, then click **`Next`**.
    4. Define the Maven project:
        - **`Group Id`**: **`com.day.cq5.myapp`**
        - **`Artifact Id`**: **`ui`**
        - **`Name`**: **`CQ5 MyApp UI`**
        - **`Description`**: **`This is the UI module`**
    5. Click **`Finish`**.
3. Set the Java Compiler to version 1.5:
    1. Right-click the `ui` project, select **`Properties`**.
    2. Select **`Java Compiler`** and set following properties to 1.5:
        - **`Compiler compliance level`**
        - **`Generated .class files compatibility`**
        - **`Source compatibility`**
    3. Click **`OK`**.
    4. In the dialog window, click **`Yes`**.
4. Create the `filter.xml` file which defines the content that will be exported by VLT:
    1. In Eclipse, navigate to `ui/scr/main` and create the `content` folder.

2. Under `content`, create the `META-INF` folder.

3. Under `META-INF`, create the `vault` folder.

4. Under `vault`, create the `filter.xml` file.

5. In `filter.xml`, copy the following code to `filter.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Defines which repository items are generally included
-->
<workspaceFilter vesion="1.0">
    <filter root="/apps/myApp" />
    <filter root="/etc/designs/myApp" />
</workspaceFilter>
```

6. Save the changes.

5. Check out the CQ5 content into your ui project with VLT:

    1. From the system command line, navigate to the directory holding your Eclipse workspace `<eclipse>/<workspace>/myApp/ui/src/main/content`.

    2. Execute the command: **vlt --credentials admin:admin co http://localhost:4502/crx**

       This command creates the folder `jcr_root` under `<eclipse>/<workspace>/myApp/ui/src/main/content`. This maps to the CRX root (/). Under `jcr_root` the following files and folders are created, as defined in `filter.xml`:

       - `apps/myApp`
       - `etc/designs/myApp`

       It also creates two files, `config.xml` and `settings.xml` in `<eclipse>/<workspace>/myApp/ui/src/main/content/META-INF/vault`. These are used by VLT.

6. To enable Eclipse to map the file paths used in the JSP scripts, create a link to the `apps` folder under `ui`:

    1. Right-click `ui`, select **`New`**, then **`Folder`**.

    2. In the dialog window, click **`Advanced`** and check the **`Link to folder in the file system`** box.

    3. Click **`Browse`**, then specify `<eclipse>/<workspace>/myApp/ui/src/main/content/jcr_root/apps`.

    4. Click **`OK`**.

    5. Click **`Finish`**.

7. To enable Eclipse to identify the Java classes, methods and objects used in the JSP scripts, export the needed Java libraries from the CQ5 server to your file system and reference them in the ui project.

   In this example, you will reference the following libraries:

   - `libs/cq/install` stored in the CQ5 server
   - `libs/sling/install` stored in the CQ5 server

- `libs/wcm/install` stored in the CQ5 server
- `<cq-installation-dir>/crx-quickstart/server/lib/container` stored in your file system

Proceed as follows:

1. In your file system, create a CQ5 libraries folder called `cq5libs`. This folder can be created anywhere.
2. Under `cq5libs`, create the folders: `cq`, `sling` and `wcm`.
3. From the system command line go to `.../cq5libs/cq` and execute **vlt co http://localhost:4502/crx /libs/cq/install .** to export the libraries stored under `/libs/cq/install` from the CQ5 server.
4. From the system command line go to `.../cq5libs/sling` and execute **vlt co http://localhost:4502/crx /libs/sling/install .** to export the libraries stored under `/libs/sling/install` from the CQ5 server.
5. From the system command line go to `.../cq5libs/wcm` and execute **vlt co http://localhost:4502/crx /libs/wcm/install .** to export the libraries stored under `/libs/wcm/install` from the CQ5 server.
6. In Eclipse, right-click the `ui` project, select **`Build Path`**, then **`Configure Build Path`**. In the dialog select the **`Libraries`** tab.
7. Click **`Add External JARS...`**, navigate to `.../cq5libs/cq/jcr_root`, select all the jar files and click **`Open`**.
8. Click **`Add External JARS...`**, navigate to `.../cq5libs/sling/jcr_root`, select all the jar files and click **`Open`**.
9. Click **`Add External JARS...`**, navigate to `.../cq5libs/wcm/jcr_root`, select all the jar files and click **`Open`**.
10. Click **`Add External JARS...`**, navigate to `<cq-installation-dir>/crx-quickstart/server/lib/container`, select all the jar files and click **`Open`**.
11. Click **`OK`**.

4.2.4.2 Create the Core Maven Project

To create the Core Maven project, follow the steps:

1. In Eclipse, create the Core Maven project:
    1. In the Menu bar, click **`File`**, select **`New`**, then **`Other...`** .
    2. In the dialog, expand the **`Maven`** folder, select **`Maven Project`** and click **`Next`**.
    3. Check the **`Create a simple project`** box and the **`Use default Workspace locations`** box, then click **`Next`**.
    4. Define the Maven project:
        - **`Group Id`**: **`com.day.cq5.myapp`**
        - **`Artifact Id`**: **`core`**
        - **`Name`**: **`CQ5 MyApp Core`**

        - **Description**: **This is the Core module**

    5. Click **Finish**.

2. Add the necessary plugins and dependencies to the core project:

    1. Open the `pom.xml` file under `core`.

    2. Copy-paste following code before the </project> tag:

```
  <packaging>bundle</packaging>

    <build>
        <plugins>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.5</source>
                    <target>1.5</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.felix</groupId>
                <artifactId>maven-bundle-plugin</artifactId>
                <version>1.4.3</version>
                <extensions>true</extensions>
                <configuration>
                    <instructions>
                        <Export-Package> com.day.cq5.myapp.*;version=
${pom.version}
                        </Export-Package>
                    </instructions>
                </configuration>
            </plugin>

        </plugins>

    </build>

    <dependencies>

        <dependency>
            <groupId>com.day.cq.wcm</groupId>
            <artifactId>cq-wcm-api</artifactId>
            <version>5.1.20</version>
        </dependency>

        <dependency>
            <groupId>com.day.cq</groupId>
            <artifactId>cq-commons</artifactId>
            <version>5.1.18</version>
        </dependency>

        <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.api</artifactId>
            <version>2.0.3-incubator-R708951</version>
        </dependency>

    </dependencies>
```

    3. Save the changes.

3. Deploy the CQ5 specific artifacts as defined in the pom.xml (`cq-wcm-api`, `cq-commons` and `org.apache.sling.api`) to the local Maven repository:

1. From the system command line go to `<your-user-dir>/.m2/repository/com/day/cq/wcm/cq-wcm-api/5.1.20` (create the folders if they don't exist) and execute **vlt co http://localhost:4502/crx /libs/wcm/install/cq-wcm-api-5.1.20.jar .** to export the library from the CQ5 server.
2. From the system command line go to `<your-user-dir>/.m2/repository/com/day/cq/cq-commons/5.1.18` (create the folders if they don't exist) and execute **vlt co http://localhost:4502/crx /libs/cq/install/cq-commons-5.1.18.jar .** to export the library from the CQ5 server.
3. From the system command line go to `<your-user-dir>/.m2/repository/org/apache/sling/org.apache.sling.api/2.0.3-incubator-R708951` (create the folders if they don't exist) and execute **vlt co http://localhost:4502/crx /libs/sling/install/org.apache.sling.api-2.0.3-incubator-R708951.jar .** to export the library from the CQ5 server.

Note

You don't need to perform this step if the three CQ5 artifacts are globally deployed for the project on a Maven repository (e.g. using Apache Archiva).

4. Set the Java Compiler to version 1.5:
   1. Right-click the `core` project, select **`Properties`**.
   2. Select **`Java Compiler`** and set following properties to 1.5:
      - **`Compiler compliance level`**
      - **`Generated .class files compatibility`**
      - **`Source compatibility`**
   3. Click **`OK`**.
   4. In the dialog window, click **`Yes`**.
5. Create the package `com.day.cq5.myapp` that will contain the Java classes under `core/src/main/java`:
   1. Under `core`, right-click `src/main/java`, select **`New`**, then **`Package`**.
   2. Name it **`com.day.cq5.myapp`** and click **`Finish`**.

4.2.5 Scripting with Eclipse and CQ5

When editing UI code use the following sequence:

- Create a template and a component with the CRX Explorer.
- Update the changes with VLT (export from the repository to your file system) .
- Create a component script (JSP) with Eclipse.
- Check in the changes from the file system into the repository with VLT.

The following example illustrates this process:

1. Create a new template with the CRX Explorer:
   1. In the CRX Explorer, under `/apps/myApp/templates`, create a new template: **`Name`**: **`contentpage`** **`Type`**: **`cq:Template`**

2. Under the `contentpage` Node, edit the Property **`jcr:title`** and add as **`Value`**: **`MyApp Content Page Template`**

3. Under the `contentpage` Node, add a new Node: **`Name`**: **`jcr:content`** **`Type`**: **`cq:PageContent`**

4. Under the `jcr:content` Node, edit the Property **`sling:resourceType`** and add as **`Value`**: **`myApp/components/contentpage`**

5. Under the `jcr:content` Node, add a new Property: **`Name`**: **`personName`** **`Value`**: **`myName`**

2. Create a new component with the CRX Explorer:

   • In the CRX Explorer, under `/apps/myApp/components`, create a new component: **`Name`**: **`contentpage`** **`Type`**: **`cq:Component`**

3. Use VLT to update the changes made from your repository to your file system, and therefore Eclipse:

   1. From the system command line navigate to `<eclipse>/<workspace>/myApp/ui/src/main/content/jcr_root`.

   2. Execute: **vlt st --show-update** to see the changes made on the repository.

   3. Execute: **vlt up** to update the changes from the repository to your file system.

4. Create the component script (JSP) with Eclipse:

   1. In Eclipse, navigate to `ui/src/main/content/jcr_root/apps/myApp/components/contentpage`.

   2. Right-click `contentpage`, select **`New`**, then **`File`**.

   3. In the dialog, name the file **`contentpage.jsp`** and click **`Finish`**.

   4. Copy the following code into `contentpage.jsp`:

```
This is the contentpage component.
```

   5. Save the changes.

5. With VLT check in the changes from the file system into the repository:

   1. From the system command line navigate to `<eclipse>/<workspace>/myApp/ui/src/main/content/jcr_root`.

   2. Execute: **vlt st** to see the changes made on the file system.

   3. Execute: **vlt add apps/myApp/components/contentpage/contentpage.jsp** to add the `contentpage.jsp` file to VLT control.

   4. Execute: **vlt ci** to commit the `contentpage.jsp` file to the repository.

6. From CQ5 create a page based on this template. Open the page to make sure it displays the following message:

```
This is the contentpage component.
```

Tip

It is possible to define the VLT commands as External Tools in Eclipse. This enables you to run the VLT commands from within Eclipse.

4.2.6 Java Developing with Eclipse and CQ5

When editing Core code use the following sequence:

- Create a Java class.
- Compile the Java class.
- Reference the jar file in the ui library.
- Embed the Java Class logic into the JSP script.
- Use VLT to check these changes to the JSP script (in the file system) into the repository.
- Use VLT to deploy the jar file (with the compiled class) from the file system into the repository.

The following example illustrates this process:

1. Create the Java class:
    1. In Eclipse, under `core/src/main/java`, right-click the `com.day.cq5.myapp` package, select **`New`**, then **`Class`**.
    2. In the dialog window, name the Java Class **`HelloPerson`** and click **`Finish`**. Eclipse creates and opens the file `HelloPerson.java`.
    3. In `HelloPerson.java` replace the existing code with the following:

```
package com.day.cq5.myapp;

import com.day.cq.wcm.api.Page;

public class HelloPerson {

private Page personPage;

public static final String PN_PERSON_NAME = "personName";

public HelloPerson(Page personPage) {
this.personPage = personPage;
}

public String getHelloMessage() {
String personName = personPage.getProperties().get(PN_PERSON_NAME).toString();

return personName != null ? personName : "--empty--";
}

}
```

    4. Save the changes.
2. Compile the Java class:
    1. Right-click the `core` project, select **`Run As`**, then **`Maven Install`**.
    2. Make sure that a new file `core-0.0.1-SNAPSHOT.jar` (containing the compiled class) is created under `core/target`.
3. Reference this jar file in the ui library to enable the code completion when accessing this class with the JSP script:

1. In Eclipse, right-click the `ui` project, select **`Build Path`**, then **`Configure Build Path`**. In the dialog select the **`Libraries`** tab.
2. Click **`Add JARS...`** and navigate to `core/target`, select the `core-0.0.1-SNAPSHOT.jar` file and click **`OK`**.
3. Click **`OK`** to close the dialog.

4. Embed the Java Class logic into the JSP script:
   1. In Eclipse, open the JSP script `contentpage.jsp` in `ui/src/main/content/jcr_root/apps/myApp/components/contentpage`.
   2. Replace the existing code with the following:

```
<%@ page import="com.day.cq5.myapp.HelloPerson" %>
<%@include file="/libs/wcm/global.jsp"%>
<%
HelloPerson hello = new HelloPerson(currentPage);
String msg = hello.getHelloMessage();
%>
Hello, <%= msg %>.</br></br>
This is the contenpage component.
```

   3. Save the changes.
5. With VTL check in the changes to the JSP script from the file system to the repository:
   1. From the system command line navigate to `<eclipse>/<workspace>/myApp/ui/src/main/content/jcr_root`.
   2. Execute: **vlt st** to see the changes made on the file system.
   3. Execute: **vlt ci** to commit the modified `contentpage.jsp` file to the repository.
6. Deploy the jar file containing the compiled class from the file system into the repository with VLT:
   1. In Eclipse, under `core/target`, copy the `core-0.0.1-SNAPSHOT.jar` file.
   2. In Eclipse navigate to `ui/scr/main/content/jcr_root/apps/myapp/install` and paste the copied file.
   3. From the system command line navigate to `<eclipse>/<workspace>/myApp/ui/src/main/content/jcr_root`.
   4. Execute: **vlt st** to see the changes made on the file system.
   5. Execute: **vlt add apps/myApp/install/core-0.0.1-SNAPSHOT.jar** to add the jar file to VLT control.
   6. Execute: **vlt ci** to commit the jar file to the repository.
7. In your browser, refresh the CQ5 page to make sure it displays following message:

```
Hello, myName.

This is the contentpage component.
```

8. In CRX Explorer, change the value myName and make sure that the new value is displayed when you refresh the page.

4.2.7 Building collaborative and automated projects

This section points to three open source softwares which enhance the development of CQ5 projects by adding collaboration and automation features:

- Subversion (SVN) to manage a central repository where all the developers involved in the project can commit and retrieve the code and the content they generate on their local instance.
- Apache Archiva to centrally store and retrieve the project libraries.
- Apache Continuum to automate the build process.

4.2.7.1 Collaboration with Subversion (SVN)

As the CQ5 repository can be mapped to your file system with VLT, it is now easy to set up a central repository with SVN. This is used by all developers in the project as a place they can commit, and retrieve, the code and content they generate on their local instances.

The setup of an SVN server is not covered in this document as many tutorials are already available online.

When VLT is in operation it creates .vlt files within the local directory structure. These .vlt files hold timestamps and other VLT-specific information that should not be checked into the SVN repository. This can be prevented by adding .vlt to the ignore list of the local SVN setup. To do this you add the following code to the local SVN setup file - settings.xml in the .subversion directory of your user's HOME directory:

```
[miscellany]
### Set global-ignores to a set of whitespace-delimited globs
### which Subversion will ignore in its 'status' output, and
### while importing or adding files and directories.
global-ignores = .vlt
```

4.2.7.2 Central dependency management with Apache Archiva

Java libraries, called artifacts in Maven language, can be managed centrally through Apache Archiva, an artifact repository that is used to store and retrieve the project artifacts. The setup of Archiva is well detailed online and can be referenced during setup. The Archiva server requires little management outside that of configuring local developer accounts to obtain access to the snapshots and full releases.

4.2.7.3 Build automation with Apache Continuum

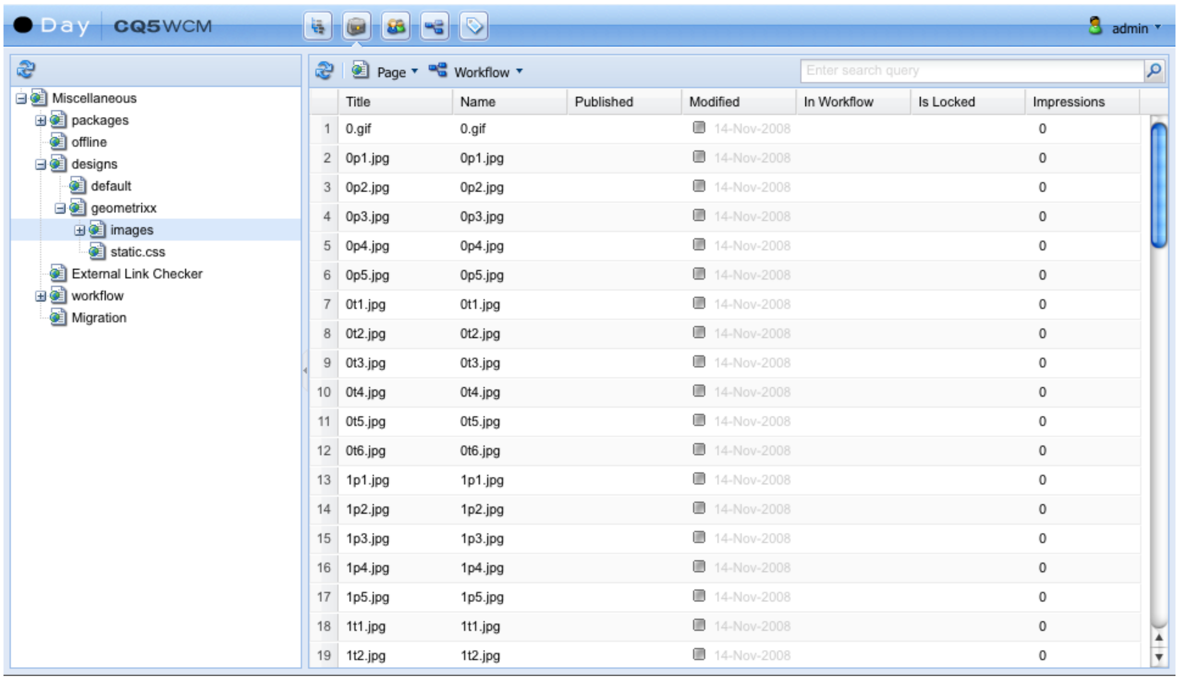
Once the project content and code is centrally available through an SVN server, it is possible to automate the build process and run the build on a daily basis (for example a nightly build). This is done with Apache Continuum, a continuous integration server with the sole duty of providing build management of artifacts and releases.

The setup of Continuum is also well detailed online.

5 Designer

You will need a design to define for your website.

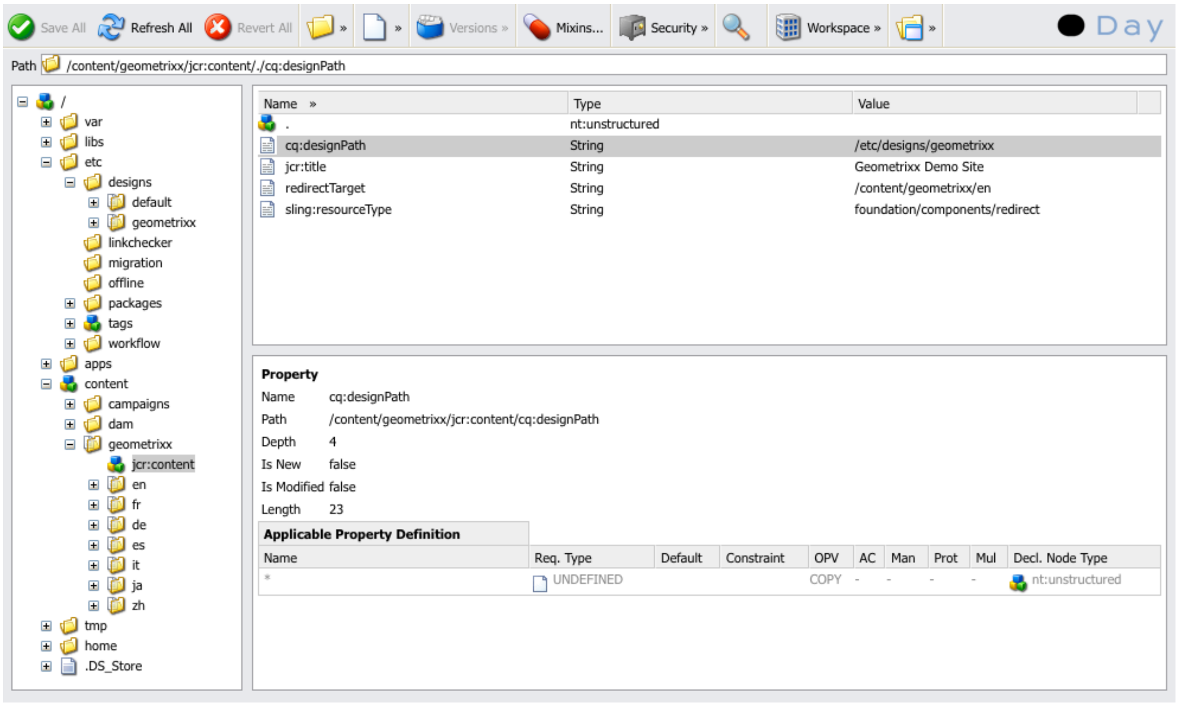
Your design can be defined in the **`designs`** section of the **`Miscellaneous`** tab:



	Title	Name	Published	Modified	In Workflow	Is Locked	Impressions
1	0.gif	0.gif		14-Nov-2008			0
2	0p1.jpg	0p1.jpg		14-Nov-2008			0
3	0p2.jpg	0p2.jpg		14-Nov-2008			0
4	0p3.jpg	0p3.jpg		14-Nov-2008			0
5	0p4.jpg	0p4.jpg		14-Nov-2008			0
6	0p5.jpg	0p5.jpg		14-Nov-2008			0
7	0t1.jpg	0t1.jpg		14-Nov-2008			0
8	0t2.jpg	0t2.jpg		14-Nov-2008			0
9	0t3.jpg	0t3.jpg		14-Nov-2008			0
10	0t4.jpg	0t4.jpg		14-Nov-2008			0
11	0t5.jpg	0t5.jpg		14-Nov-2008			0
12	0t6.jpg	0t6.jpg		14-Nov-2008			0
13	1p1.jpg	1p1.jpg		14-Nov-2008			0
14	1p2.jpg	1p2.jpg		14-Nov-2008			0
15	1p3.jpg	1p3.jpg		14-Nov-2008			0
16	1p4.jpg	1p4.jpg		14-Nov-2008			0
17	1p5.jpg	1p5.jpg		14-Nov-2008			0
18	1t1.jpg	1t1.jpg		14-Nov-2008			0
19	1t2.jpg	1t2.jpg		14-Nov-2008			0

Here you can create the structure required to store the design and upload the cascaded style sheets and images required.

Designs are stored under `/etc/designs`. The path to the design to be used for a website is specified using the `cq:designPath` property of the `jcr:content` node.



Path /content/geometrixx/jcr:content/./cq:designPath

Name	Type	Value
.	nt:unstructured	
cq:designPath	String	/etc/designs/geometrixx
jcr:title	String	Geometrixx Demo Site
redirectTarget	String	/content/geometrixx/en
sling:resourceType	String	foundation/components/redirect

Property

Name	cq:designPath
Path	/content/geometrixx/jcr:content/cq:designPath
Depth	4
Is New	false
Is Modified	false
Length	23

Applicable Property Definition

Name	Req. Type	Default	Constraint	OPV	AC	Man	Prot	Mul	Decl. Node Type
*	UNDEFINED			COPY	-	-	-	-	nt:unstructured

6 Templates

6.1 What are Templates?

A Template is used to create a Page and defines which components can be used within the selected scope. A template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content.

Each Template will present you with a selection of components available for use.

- Templates are built up of Components; components
- Components use, and allow access to, Widgets and these are used to render the Content.

6.2 Overview of templates

CQ WCM comes with several templates including a contentpage, redirect page, and home page.

Table 6.1. Templates within CQ5 (/apps/geometrixx/components and /libs/foundation/components)

Title	Component	Location	Purpose	Equivalent in CQ4
Home Page	homepage	G	The Geometrixx home page template.	
Content Page	contentpage	G	The Geometrixx content page template.	
Redirect	redirect	libs	Redirect. Component and Template.	

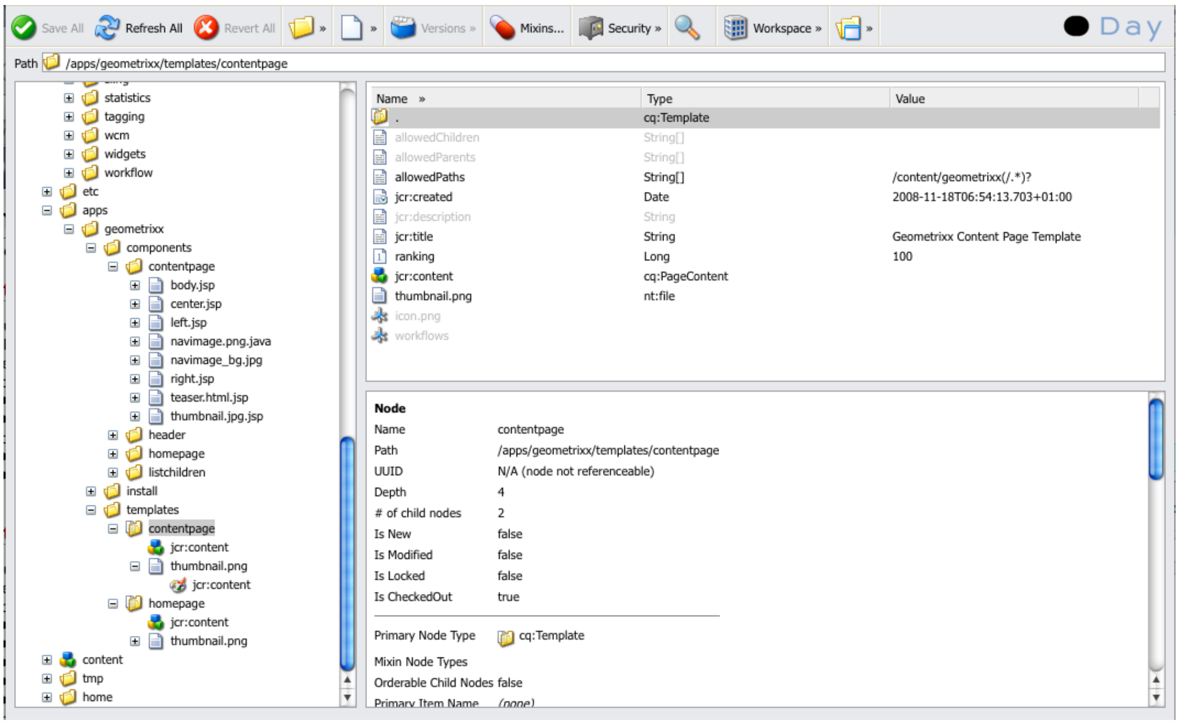
6.3 How Templates are structured

There are two aspects to be considered:

- the structure of the template itself
- the structure of the content produced when a template is used

6.3.1 The structure of a Template

A Template is created under a node of type cq:Template.



Path /apps/geometrixx/templates/contentpage

Name	Type	Value
.	cq:Template	
allowedChildren	String[]	
allowedParents	String[]	
allowedPaths	String[]	/content/geometrixx(/.*)?
jcr:created	Date	2008-11-18T06:54:13.703+01:00
jcr:description	String	
jcr:title	String	Geometrixx Content Page Template
ranking	Long	100
jcr:content	cq:PageContent	
thumbnail.png	nt:file	
icon.png		
workflows		

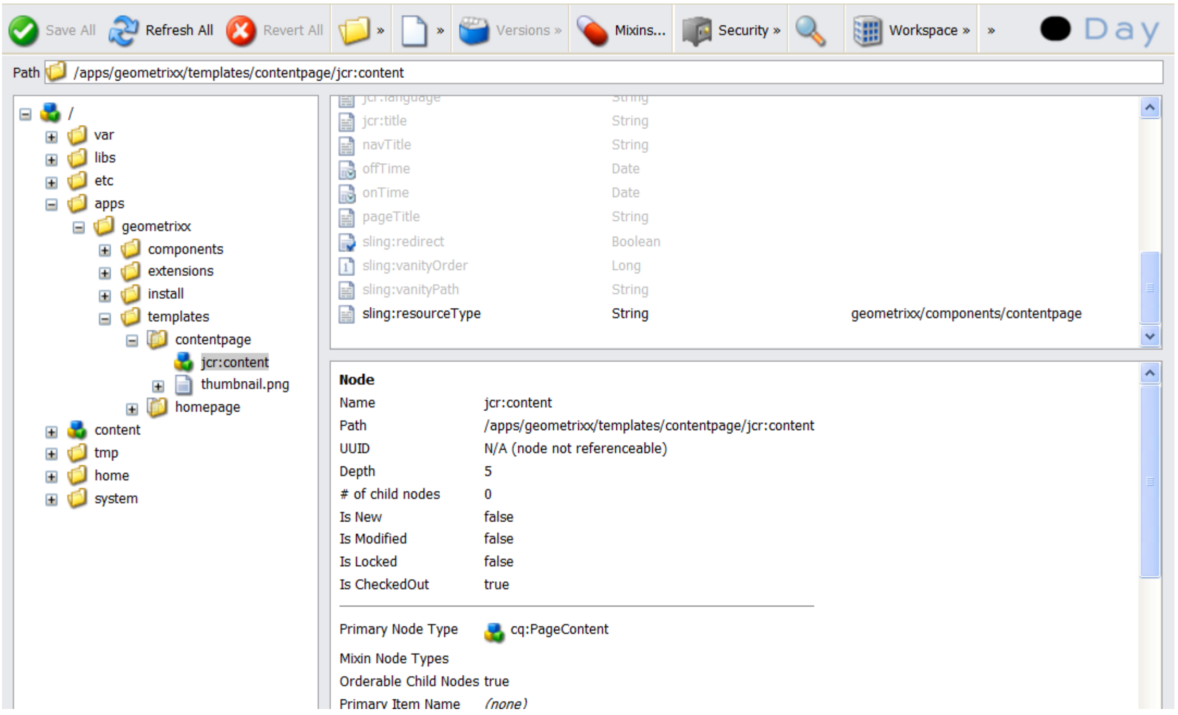
Node

Name	contentpage
Path	/apps/geometrixx/templates/contentpage
UUID	N/A (node not referenceable)
Depth	4
# of child nodes	2
Is New	false
Is Modified	false
Is Locked	false
Is CheckedOut	true
Primary Node Type	cq:Template
Mixin Node Types	
Orderable Child Nodes	false
Primary Item Name	(none)

Various properties can be set, in particular:

- *jcr:title* - title for the template; appears in the dialog when creating a page.
- *jcr:description* - description for the template; appears in the dialog when creating a page.

This node contains a jcr:Content (cq:PageContent) node which be used as the basis for the content node of resulting pages; this references, using sling:resourceType, the component to be used for rendering the actual content of a new page.



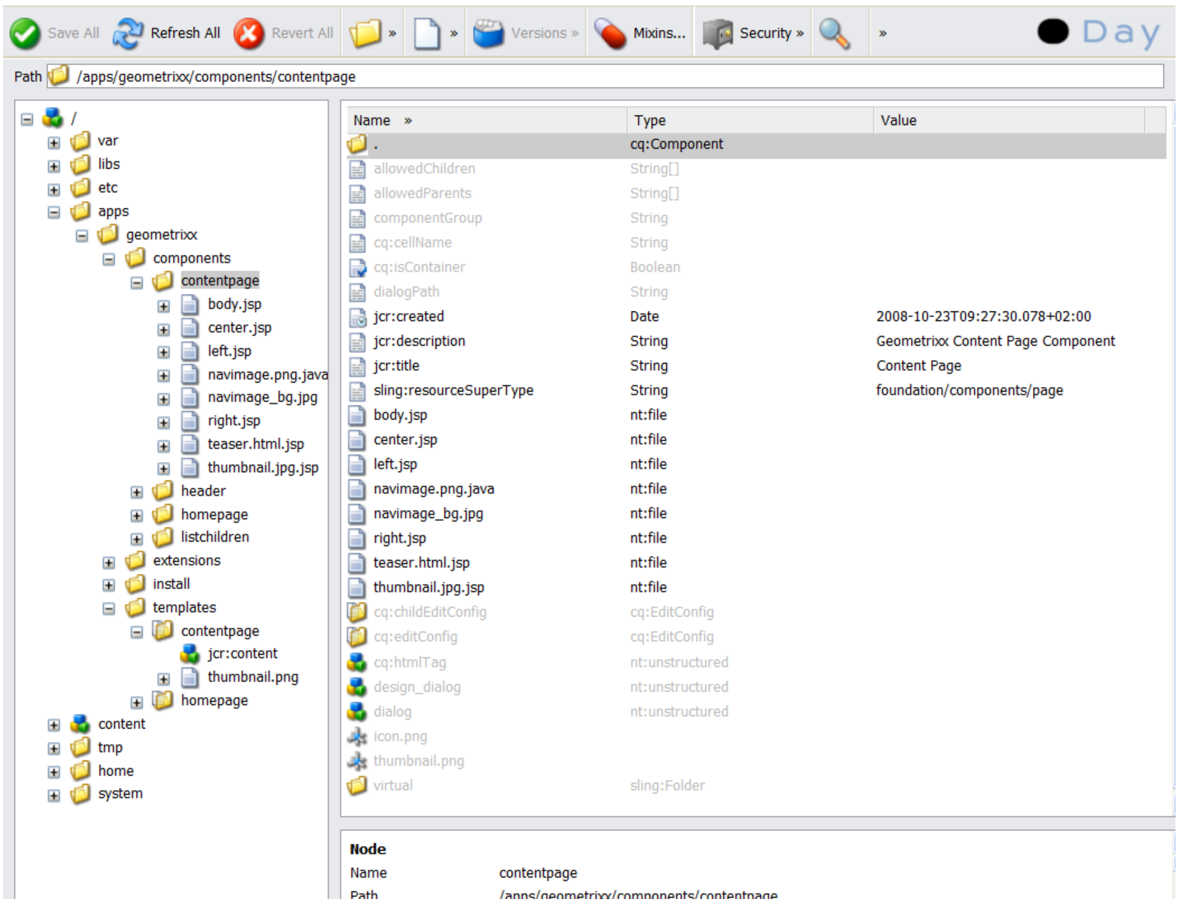
Path /apps/geometrixx/templates/contentpage/jcr:content

Name	Type	Value
jcr:language	String	
jcr:title	String	
navTitle	String	
offTime	Date	
onTime	Date	
pageTitle	String	
sling:redirect	Boolean	
sling:vanityOrder	Long	
sling:vanityPath	String	
sling:resourceType	String	geometrixx/components/contentpage

Node

Name	jcr:content
Path	/apps/geometrixx/templates/contentpage/jcr:content
UUID	N/A (node not referenceable)
Depth	5
# of child nodes	0
Is New	false
Is Modified	false
Is Locked	false
Is CheckedOut	true
Primary Node Type	cq:PageContent
Mixin Node Types	
Orderable Child Nodes	true
Primary Item Name	(none)

This component is used to define the structure and design of the content when a new page is created.



Path /apps/geometrixx/components/contentpage

```
/
  var
  libs
  etc
  apps
    geometrixx
      components
        contentpage
          body.jsp
          center.jsp
          left.jsp
          navimage.png.java
          navimage_bg.jpg
          right.jsp
          teaser.html.jsp
          thumbnail.jpg.jsp
        header
        homepage
        listchildren
      extensions
      install
      templates
        contentpage
          jcr:content
          thumbnail.png
        homepage
  content
  tmp
  home
  system
```

Name	Type	Value
.	cq:Component	
allowedChildren	String[]	
allowedParents	String[]	
componentGroup	String	
cq:cellName	String	
cq:isContainer	Boolean	
dialogPath	String	
jcr:created	Date	2008-10-23T09:27:30.078+02:00
jcr:description	String	Geometrixx Content Page Component
jcr:title	String	Content Page
sling:resourceSuperType	String	foundation/components/page
body.jsp	nt:file	
center.jsp	nt:file	
left.jsp	nt:file	
navimage.png.java	nt:file	
navimage_bg.jpg	nt:file	
right.jsp	nt:file	
teaser.html.jsp	nt:file	
thumbnail.jpg.jsp	nt:file	
cq:childEditConfig	cq:EditConfig	
cq:editConfig	cq:EditConfig	
cq:htmlTag	nt:unstructured	
design_dialog	nt:unstructured	
dialog	nt:unstructured	
icon.png		
thumbnail.png		
virtual	sling:Folder	

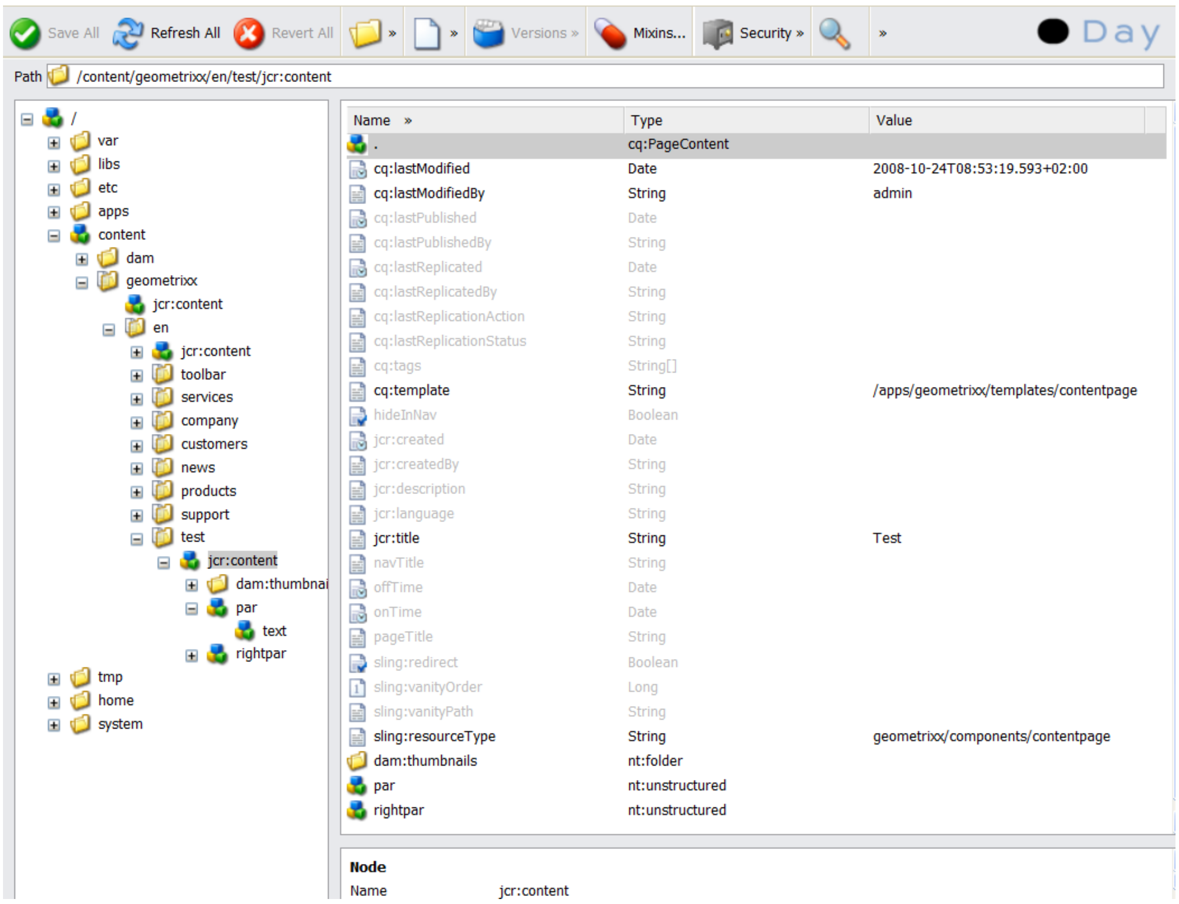
Node

Name	contentpage
Path	/apps/geometrixx/components/contentpage

6.3.2 The content produced by a Template

Templates are used to create Pages of type cq:Page (as mentioned earlier, a Page is a special type of Component). Each CQ WCMS Page has a structured node jcr:Content. This:

- is of type cq:PageContent
- is a structured node-type holding a defined content-definition
- has a property sling:resourceType to reference the component holding the sling scripts used for rendering the content



Path /content/geometrixx/en/test/jcr:content

```
/
  var
  libs
  etc
  apps
  content
    dam
    geometrixx
      jcr:content
      en
        jcr:content
        toolbar
        services
        company
        customers
        news
        products
        support
        test
          jcr:content
            dam:thumbnai
            par
              text
            rightpar
  tmp
  home
  system
```

Name »	Type	Value
.	cq:PageContent	
cq:lastModified	Date	2008-10-24T08:53:19.593+02:00
cq:lastModifiedBy	String	admin
cq:lastPublished	Date	
cq:lastPublishedBy	String	
cq:lastReplicated	Date	
cq:lastReplicatedBy	String	
cq:lastReplicationAction	String	
cq:lastReplicationStatus	String	
cq:tags	String[]	
cq:template	String	/apps/geometrixx/templates/contentpage
hideInNav	Boolean	
jcr:created	Date	
jcr:createdBy	String	
jcr:description	String	
jcr:language	String	
jcr:title	String	Test
navTitle	String	
offTime	Date	
onTime	Date	
pageTitle	String	
sling:redirect	Boolean	
sling:vanityOrder	Long	
sling:vanityPath	String	
sling:resourceType	String	geometrixx/components/contentpage
dam:thumbnails	nt:folder	
par	nt:unstructured	
rightpar	nt:unstructured	

Node
Name jcr:content

6.4 Developing Page Templates

CQ5 page templates are simply models used to create new pages. They can contain as little, or as much, initial content as needed, their role being to create the correct initial node structures, with the required properties (primarily sling:resourceType) set to allow editing and rendering.

6.4.1 Creating a new Template (based on an existing template)

Needless to say a new template can be created completely from scratch, but often an existing template will be copied and updated to save you time and effort. For example, the templates within Geometrixx can be used to get you started.

1. Copy an existing template (preferably with a definition as close as possible to what you want to achieve) to a new node.

   Note

   Templates are usually stored in `/apps/<website-name>/templates/<template-name>`.

2. Change the jcr:title of the new template node to reflect its new role. You can also update the jcr:description if appropriate.

3. Copy the component on which the template is based (this is indicated by the sling:resourceType property of the jcr:content node within the template) to create a new instance.

Note

Components are usually stored in `/apps/<website-name>/components/<component-name>`.

4. Update the jcr:title and jcr:description of the new component.

5. Replace the `thumbnail.png` if you want a new thumbnail picture to be shown in the template selection list.

6. Update the sling:resourceType of the template's jcr:content node to reference the new component.

7. Make any further changes to the functionality or design of the template and/or its underlying component.

   Changes made to the `/apps/<website>/templates/<template-name>` node will affect the template instance (as in the selection list).

   Changes made to the `/apps/<website>/components/<component-name>` node will affect the content page created when the template is used.

You can now create a page within your website using the new template.

6.5 Summary

Summary:

Location: /apps/<myapp>/templates

Root Node:

- <mytemplate> (cq:Template) - Hierarchy node of the template

Vital Properties:

- jcr:title - Template title, appears in the Create Page Dialog
- jcr:description - Template description, appears in the Create Page Dialog

Vital Child Nodes:

- jcr:content (cq:PageContent) - Content node for template instantiations

Vital Properties of Child Node jcr:content:

- sling:resourceType - Reference to the rendering component

The template is a blueprint of a specific type of page. To create a page the template must be copied (node-tree /apps/<myapp>/templates/<mytemplate>) to the corresponding position in the site-tree (this is what happens if a page is created using the siteadmin). This copy action also gives the page its initial content (usually Top-Level Content only) and the property sling:resourceType, the path to the page component that is used to render the page (everything in the child node jcr:content).

7 Components

7.1 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
- run anywhere within any CQ system
- have a standardized user interface
- use widgets

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 CQ component:

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

Components included with CQ include:

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

Note

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

7.2 Overview of components

The following components are included in the basic installation within the Geometrixx website:

Note

This table lists all components - some of which are also Templates.

Table 7.1. Components within CQ5 (/apps/geometrixx/components and /libs/foundation/components)

Title	Component	Location	Purpose	Equivalent in CQ4
Breadcrumb	breadcrumb	libs	Generates breadcrumb navigation.	breadcrumb
Chart	chart	libs	Chart, line or pie chart.	

Title	Component	Location	Purpose	Equivalent in CQ4
Column Control 2 Columns 3 Columns	parsys/colctrl	libs	Mechanism for controlling and formatting columns. 2 and 3 Columns are the same component, but default to 2 and 3 columns respectively.	columncontrol
Content Page	contentpage (template)	G	The content page template.	
Download	download	libs	Enables access to a file made available for download.	download
Flash	flash	libs	Allows you to enter a flash movie.	
Forms Address Field	form/address	libs	A complex field allowing the input of an international address.	forms
Forms Begin	form/start	libs	Begins the form definition.	forms
Forms Captcha	form/captcha	libs	A field consisting of an alphanumeric word that refreshes automatically. The captcha component protects websites against bots.	
Forms Checkbox Group	form/checkbox	libs	Multiple items organized into a list and preceded by check boxes. Users can select multiple check boxes.	
Forms Dropdown List	form/dropdown	libs	Multiple items organized into a drop-down list. The **`Multi Selectable`** switch specifies if several elements can be selected from the list.	
Forms End	form/end	libs	Terminates the form definition.	forms
Forms File Upload	form/upload	libs	An upload element that allows the user to upload a file to the server.	
Forms Hidden Field	form/hidden	libs	This field is not displayed to the user. It can be used to transport a value to the client and back to the server. This field should have no constraints.	
Forms Image Button	form/ imagebutton	libs	An additional submit button for the form that is rendered as an image.	
Forms Password Field	form/password	libs	Same as text field but only a single line is allowed and the text input from the user is not visible in the field.	
Forms Radio Group	form/radio	libs	Multiple items organized into a list preceded by a radio button. Users must select only one radio button.	
Forms Submit Button	form/submit	libs	An additional submit button for the form where the title is displayed as text on the button.	
Forms Text Field	form/text	libs	Text field that users enter information into.	

Title	Component	Location	Purpose	Equivalent in CQ4
Header	header	G	Page header used for the home page.	
Home Page	homepage	G	The home page template.	
Image	image	libs	Displays an image, with accompanying text positioned to the right.	image
Iparsys	iparsys	libs	Inherited paragraph system	
List	list	libs	Displays a configurable list of searched items.	generic list
List Children	listchildren	G	Generates a list of all pages beneath the current.	listchildren
Logo	logo	libs	Logo.	logo
	page	libs		
	parbase	libs		
	parsys	libs		
Redirect	redirect (template)	libs	Redirect. Component and Template.	
Reference	reference	libs	Reference.	
Search	search	libs	A search dialog with related search functionality.	search
Sitemap	sitemap	libs	A sitemap listing all pages.	
Slideshow	slideshow	libs	Allows you to load a slideshow.	
Table	table	libs	A table, with various formatting options.	table
Teaser	teaser	libs	A piece of content, often an image displayed on a main page intended to 'tease' users into accessing the underlying content.	
Text	text	libs	A text item.	richtexteditor
Text Image	textimage	libs	Text with an accompanying image.	textimagejcr
Title	title	libs	Title of the page (can be different from the page name)	
Toolbar	toolbar	libs	A toolbar	
Top Navigation	topnav	libs	Navigation at the top of the page.	topnavigation

Note

Location indicates the location of the component within the repository

libs = included in the standard library (`/libs/foundation/components`)

G = included with Geometrixx (`/apps/geometrixx/components`)

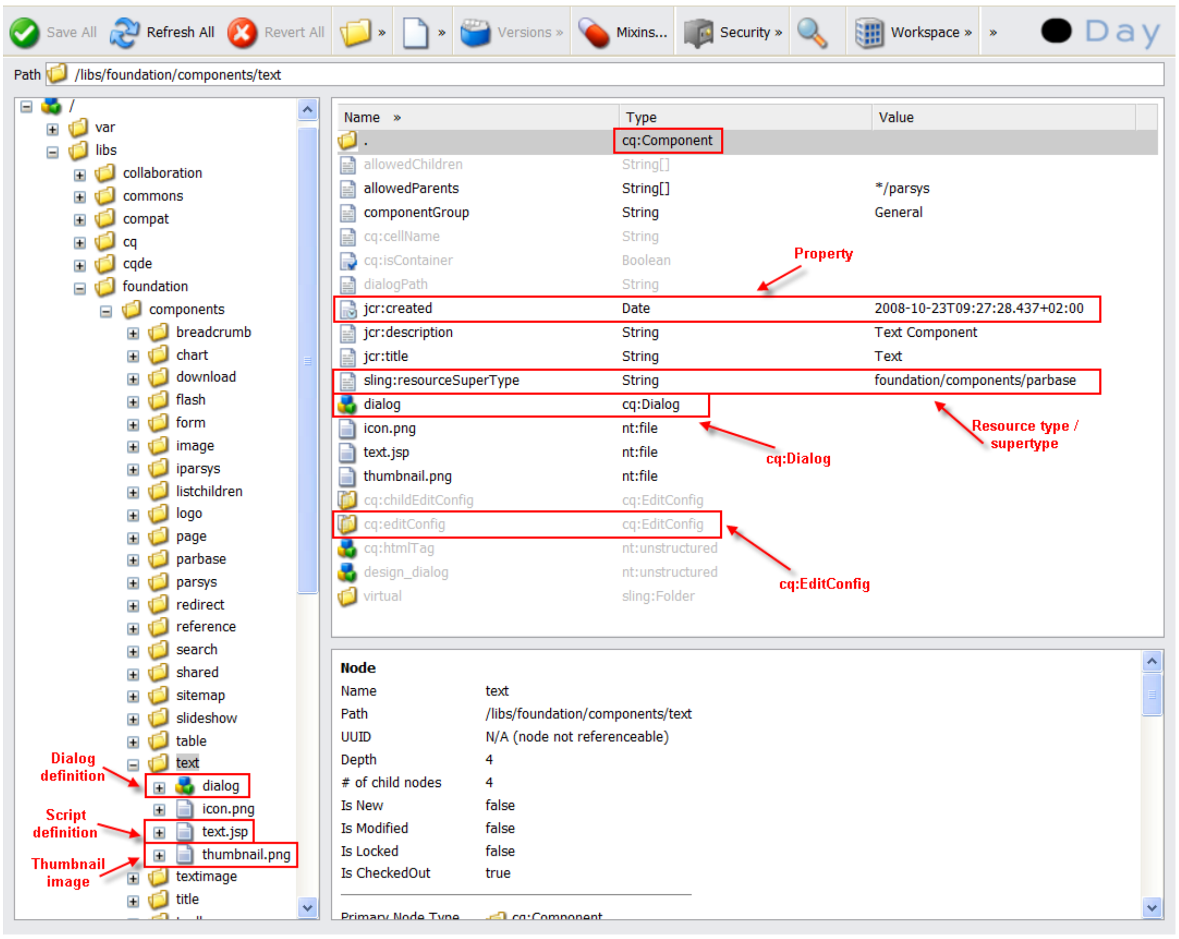
7.3 Components and their structure

7.3.1 Component definitions

As already mentioned, the definition of a component can be broken down into:

- CQ components are based on Sling
- CQ WCM components are located under `/libs/foundation/components`
- site specific components are located under `/apps/<sitename>/components`
- CQ WCM has the **page** component; this is a particular type of resource which is important for content management.
- CQ5 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 `dialog` (of type `cq:Dialog`) and `cq:editConfig` (of type `cq:EditConfig`) must both be defined as otherwise the component will not appear
  - *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:



Path /libs/foundation/components/text

```
/
  var
  libs
    collaboration
    commons
    compat
    cq
    cqde
    foundation
      components
        breadcrumb
        chart
        download
        flash
        form
        image
        iparsys
        listchildren
        logo
        page
        parbase
        parsys
        redirect
        reference
        search
        shared
        sitemap
        slideshow
        table
        text
          dialog
          icon.png
          text.jsp
          thumbnail.png
        textimage
        title
```

Name »	Type	Value
.	cq:Component	
allowedChildren	String[]	
allowedParents	String[]	*/parsys
componentGroup	String	General
cq:cellName	String	
cq:isContainer	Boolean	
dialogPath	String	
jcr:created	Date	2008-10-23T09:27:28.437+02:00
jcr:description	String	Text Component
jcr:title	String	Text
sling:resourceSuperType	String	foundation/components/parbase
dialog	cq:Dialog	
icon.png	nt:file	
text.jsp	nt:file	
thumbnail.png	nt:file	
cq:childEditConfig	cq:EditConfig	
cq:editConfig	cq:EditConfig	
cq:htmlTag	nt:unstructured	
design_dialog	nt:unstructured	
virtual	sling:Folder	

Node	
Name	text
Path	/libs/foundation/components/text
UUID	N/A (node not referenceable)
Depth	4
# of child nodes	4
Is New	false
Is Modified	false
Is Locked	false
Is CheckedOut	true
Primary Node Type	cq:Component

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
- *`thumbnail.png`* - graphics file to be used as a thumbnail for the component

Child nodes of particular interest include:

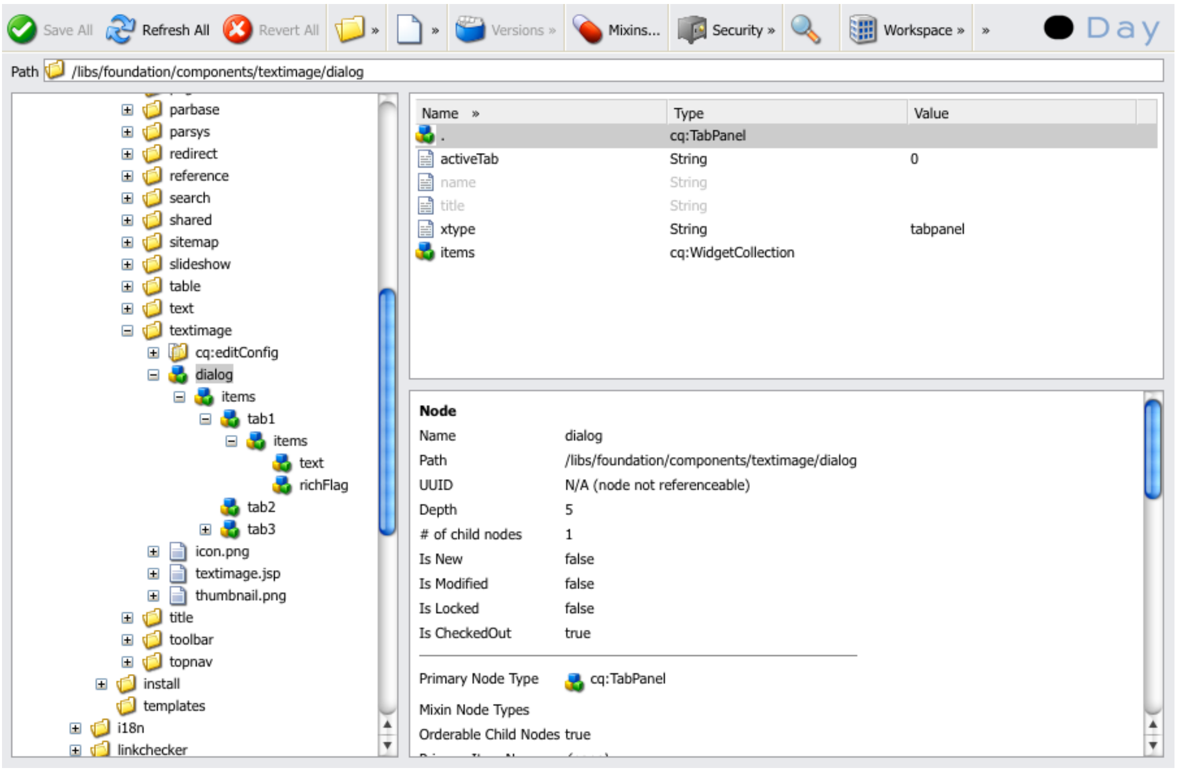
- *`cq:editConfig (cq:EditConfig)`* - this controls visual aspects; for example, it can define the appearance of a bar or widget, or can add customized controls
- *`cqchildEditConfig(cq:EditConfig)`* - this controls the visual aspects for child components that do not have their own definitions
- *`dialog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

7.3.1.1 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`*:



Path /libs/foundation/components/textimage/dialog

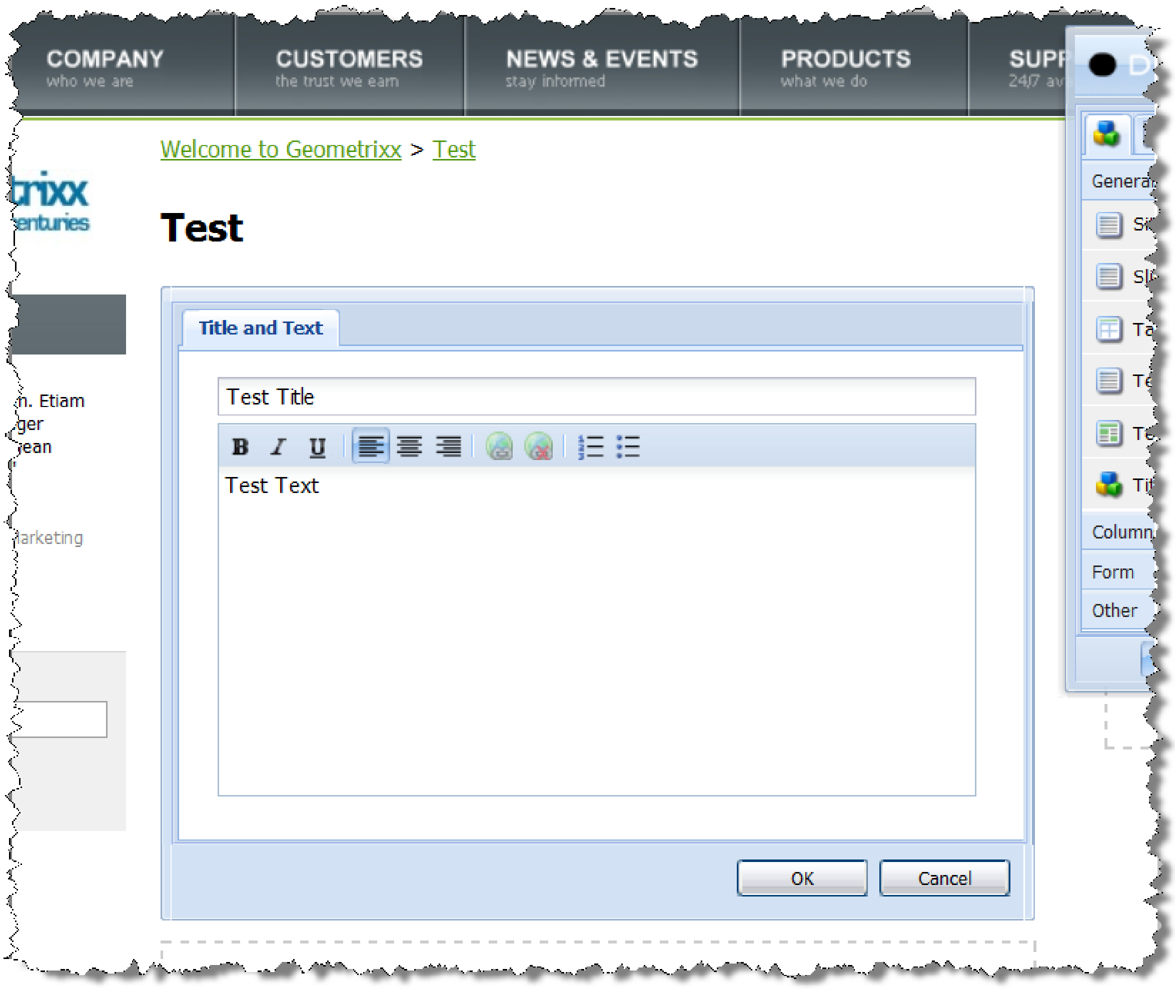
Name »	Type	Value
.	cq:TabPanel	
activeTab	String	0
name	String	
title	String	
xtype	String	tabpanel
items	cq:WidgetCollection	

Node	
Name	dialog
Path	/libs/foundation/components/textimage/dialog
UUID	N/A (node not referenceable)
Depth	5
# of child nodes	1
Is New	false
Is Modified	false
Is Locked	false
Is CheckedOut	true
Primary Node Type	cq:TabPanel
Mixin Node Types	
Orderable Child Nodes	true

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.

7.3.2 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:

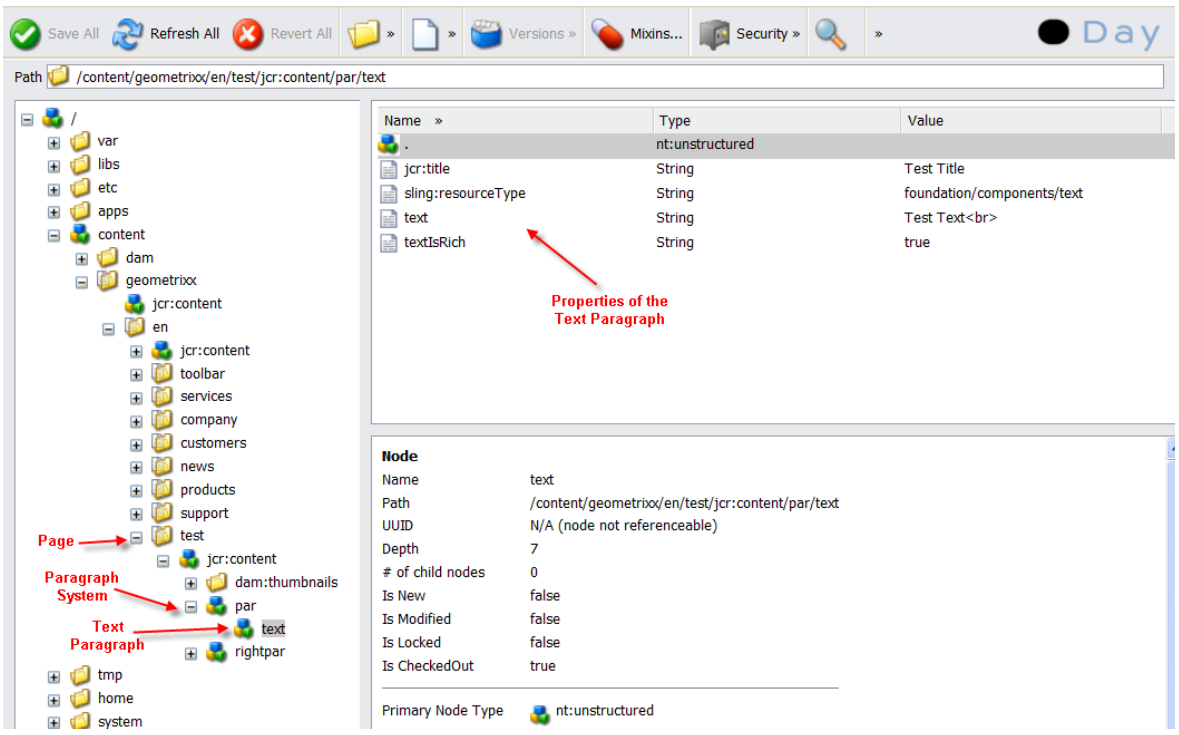


Test

Test Title

Test Text

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



Path /content/geometrixx/en/test/jcr:content/par/text

```
/
  var
  libs
  etc
  apps
  content
    dam
    geometrixx
      jcr:content
      en
        jcr:content
        toolbar
        services
        company
        customers
        news
        products
        support
        test
          jcr:content
            dam:thumbnails
            par
              text
            rightpar
  tmp
  home
  system
```

Name »	Type	Value
.	nt:unstructured	
jcr:title	String	Test Title
sling:resourceType	String	foundation/components/text
text	String	Test Text
textIsRich	String	true

Node

Name	text
Path	/content/geometrixx/en/test/jcr:content/par/text
UUID	N/A (node not referenceable)
Depth	7
# of child nodes	0
Is New	false
Is Modified	false
Is Locked	false
Is CheckedOut	true

Primary Node Type nt:unstructured

In particular, if you look at the title:

- the definition of `/libs/foundation/components/text/dialog/items/title` has the property name=./jcr:title
- within the content, this generates the property jcr:title holding the input **`Test Title`**.

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

7.3.3 Component Hierarchy and Inheritance

Components within CQ 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.

7.3.4 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/thumbnail.png - Icon & thumbnail 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.

7.4 Developing Components

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

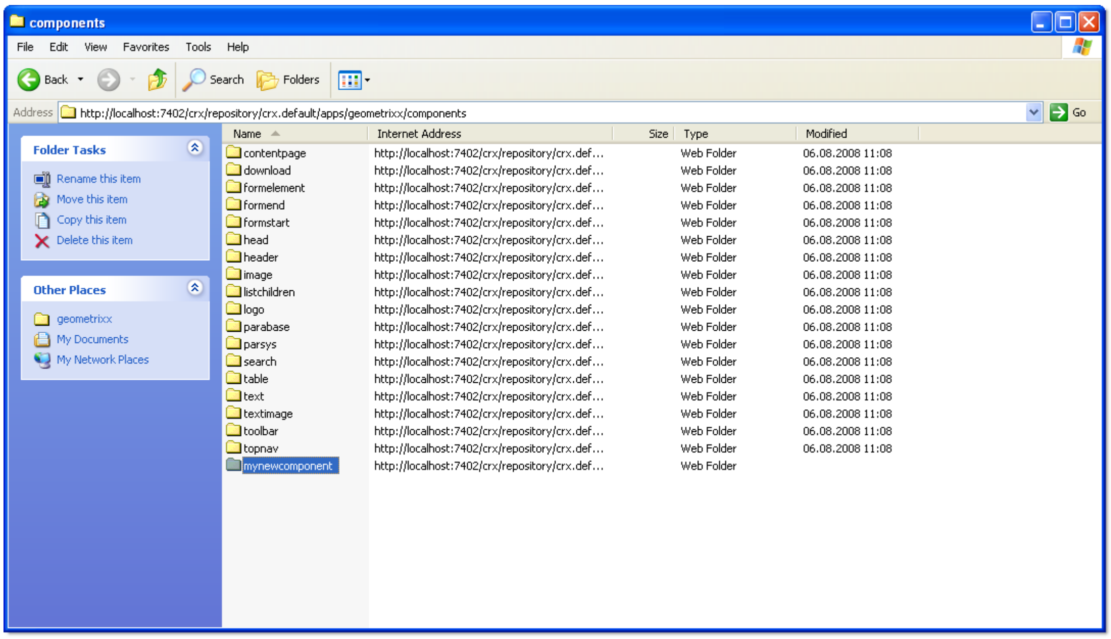
A quick way to get started is to copy an existing component and then make the changes you want. You can also use this method to edit existing components (although Day recommends that you back up the original component).

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

7.4.1 Developing a new component by adapting an existing component

To develop new components for CQ WCM based on existing component, you copy the component and create a javascript file for the new component and store it in a location accessible to CQ5:

1. Create a new component folder in `/apps/<website-name>/components/<MyComponent>` by copying an existing component, such as the Text component, and renaming it.



components

Address http://localhost:7402/crx/repository/crx.default/apps/geometrixx/components

Name	Internet Address	Size	Type	Modified
contentpage	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
download	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
formelement	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
formend	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
formstart	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
head	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
header	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
image	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
listchildren	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
logo	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
parabase	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
parsys	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
search	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
table	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
text	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
textimage	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
toolbar	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
topnav	http://localhost:7402/crx/repository/crx.def...		Web Folder	06.08.2008 11:08
mynewcomponent	http://localhost:7402/crx/repository/crx.def...		Web Folder	

2. In the CRX Explorer, modify the `jcr:description` and `jcr:title` to reflect its new name.

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. In the Content Explorer, navigate to the component and change the `allowedParents` property to `*/parsys`, which makes it available to the paragraph system.

   Note

   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 `/apps/<website-name>/components/<MyComponent>` to the `/etc/designs/default/<website-name>/jcr:content/contentpage/parsys/components` property in CRX or by following the instructions in Adding new components to paragraph systems.

6. In CQ 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.

7.4.2 Adding a new component to the paragraph system (design mode)

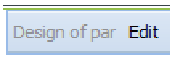
After the component has been developed, you add it to the paragraph system, which enables authors to select and use the component when editing a page.

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 `?cmsmode=design` to the end of the URL and accessing again; for example `<contextPath>/ Test.html?cmsmode=design`.
   - clicking **`Design`** in Sidekick

   You are now in designmode 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.

7.4.3 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.

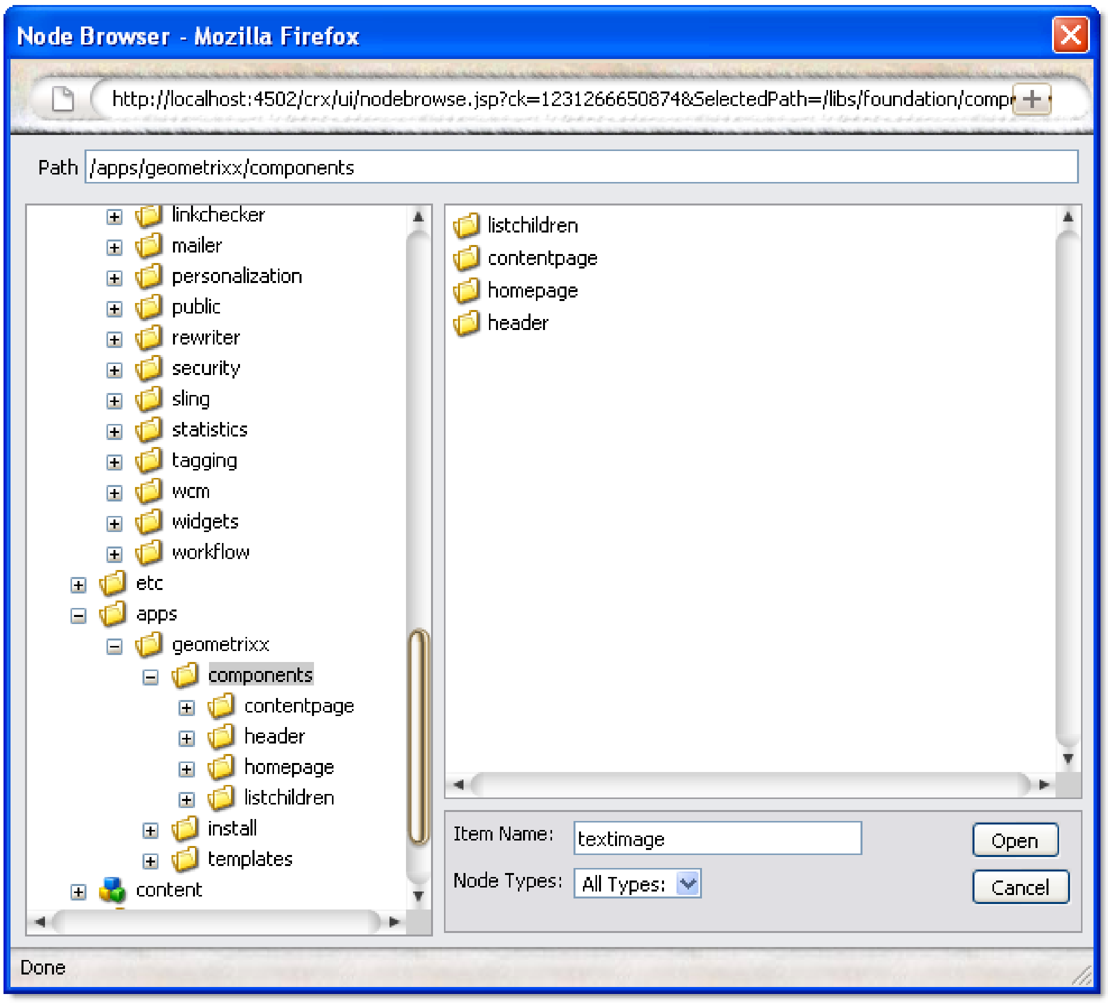
The following techniques are described in this exercise:

- 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

7.4.3.1 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 CQ WCM example application. To extend the textimage component, go to the CRX Explorer (*`server name:port number`*`/crx`) and log in as **`admin`** and then navigate to the Content Explorer.

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.)



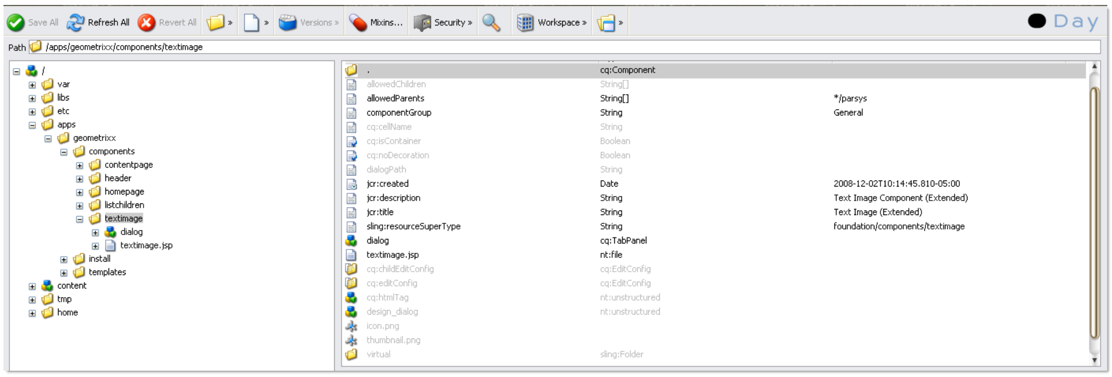
2. To keep this example simple, navigate to the component you copied and delete all the subnodes of the new textimage node **`except`** for the following ones:

   - dialog definition: `textimage/dialog`
   - component script: `textimage/textimage.jsp`

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 these steps the component node looks like the following:



Path /apps/geometrixx/components/textimage

```
/
  var
  libs
  etc
  apps
    geometrixx
      components
        contentpage
        header
        homepage
        listchildren
        textimage
          dialog
          textimage.jsp
      install
      templates
  content
  tmp
  home
```

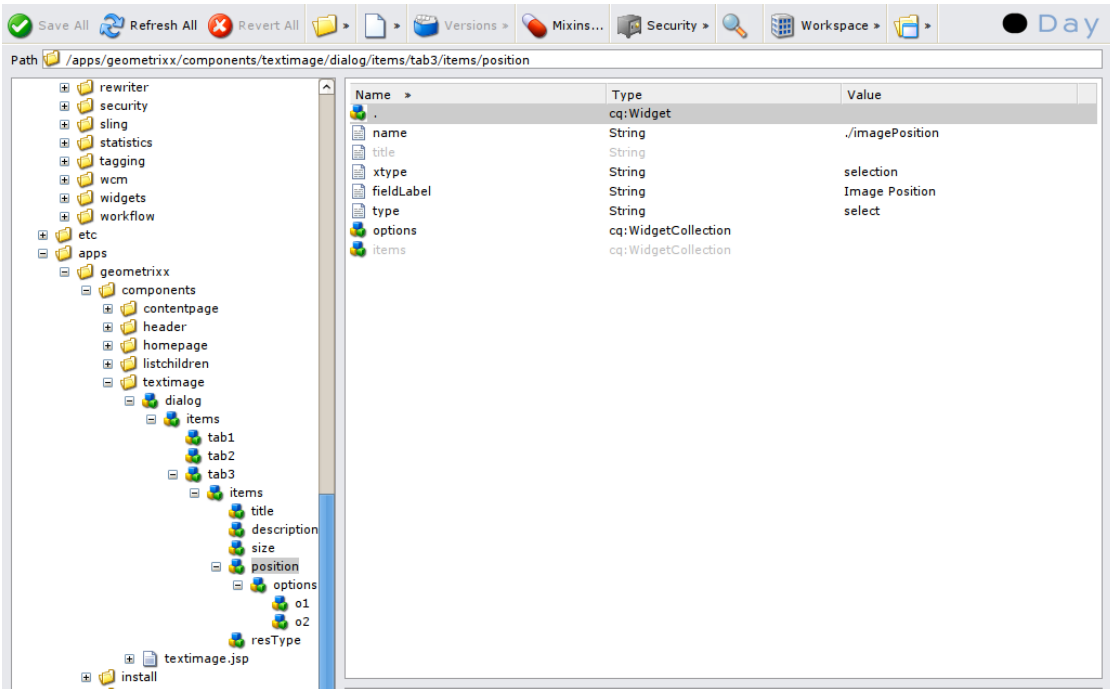
.	cq:Component	
allowedChildren	String[]	
allowedParents	String[]	*/parsys
componentGroup	String	General
cq:cellName	String	
cq:isContainer	Boolean	
cq:noDecoration	Boolean	
dialogPath	String	
jcr:created	Date	2008-12-02T10:14:45.810-05:00
jcr:description	String	Text Image Component (Extended)
jcr:title	String	Text Image (Extended)
sling:resourceSuperType	String	foundation/components/textimage
dialog	cq:TabPanel	
textimage.jsp	nt:file	
cq:childEditConfig	cq:EditConfig	
cq:editConfig	cq:EditConfig	
cq:htmlTag	nt:unstructured	
design_dialog	nt:unstructured	
icon.png		
thumbnail.png		
virtual	sling:Folder	

4. 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 three subnodes, `tab1` to `tab3`, representing the three 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 *`pathParameter`* 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`*

   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:



Path /apps/geometrixx/components/textimage/dialog/items/tab3/items/position

```
rewriter
security
sling
statistics
tagging
wcm
widgets
workflow
etc
apps
  geometrixx
    components
      contentpage
      header
      homepage
      listchildren
      textimage
        dialog
          items
            tab1
            tab2
            tab3
              items
                title
                description
                size
                position
                  options
                    o1
                    o2
                resType
        textimage.jsp
    install
```

Name	Type	Value
.	cq:Widget	
name	String	./imagePosition
title	String	
xtype	String	selection
fieldLabel	String	Image Position
type	String	select
options	cq:WidgetCollection	
items	cq:WidgetCollection	

5. Extend the component script, `textimage.jsp`, with extra handling of the new parameter.

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

```
    Image img = new Image(resource, "image");
    if (img.hasContent() || WCMMode.fromRequest(request) == WCMMode.EDIT) {
%><div class="image"><%
        img.loadStyleData(currentStyle);
```

  We are going to replace the emphasized code fragment *`%><div class="image"><%`* with new code generating a custom style for this tag.

- Copy the following code fragment, and replace the *`%><div class="image"><%`* 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"><%
```

  Note that 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.

- The code fragment, after the change, should look like this (new code emphasized):

```
    Image img = new Image(resource, "image");
    if (img.hasContent() || WCMMode.fromRequest(request) == WCMMode.EDIT) {
        // 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"><%
        img.loadStyleData(currentStyle);
```

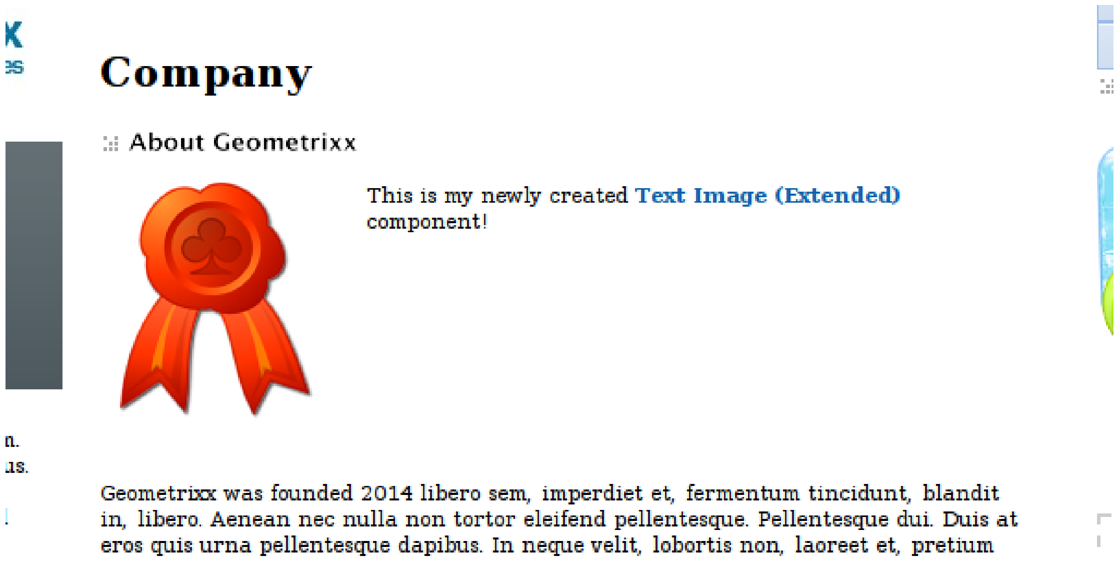
- Save the script to the repository.

6. The component is ready to test.

7.4.3.2 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 and you should see the default rendering of Text and Image component:



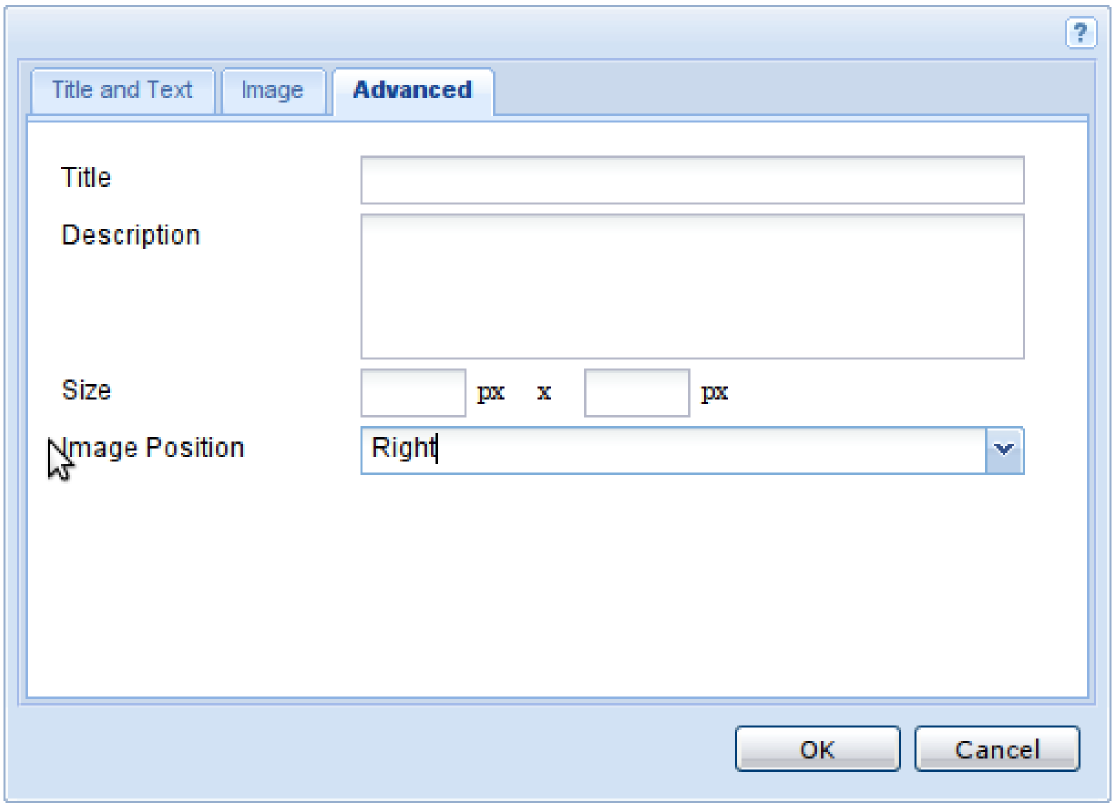
Company

About Geometrixx

This is my newly created Text Image (Extended) component!

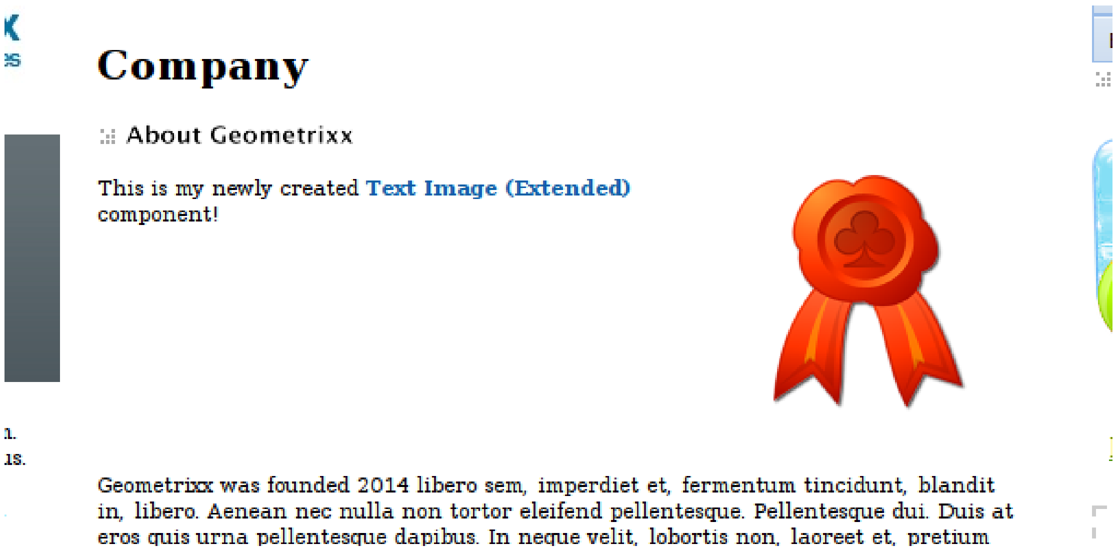
Geometrixx was founded 2014 libero sem, imperdiet et, fermentum tincidunt, blandit in, libero. Aenean nec nulla non tortor eleifend pellentesque. Pellentesque dui. Duis at eros quis urna pellentesque dapibus. In neque velit, lobortis non, laoreet et, pretium

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.



Title	
Description	
Size	px x px
Image Position	Right

7. You see the paragraph rendered with the image on the right:



Company

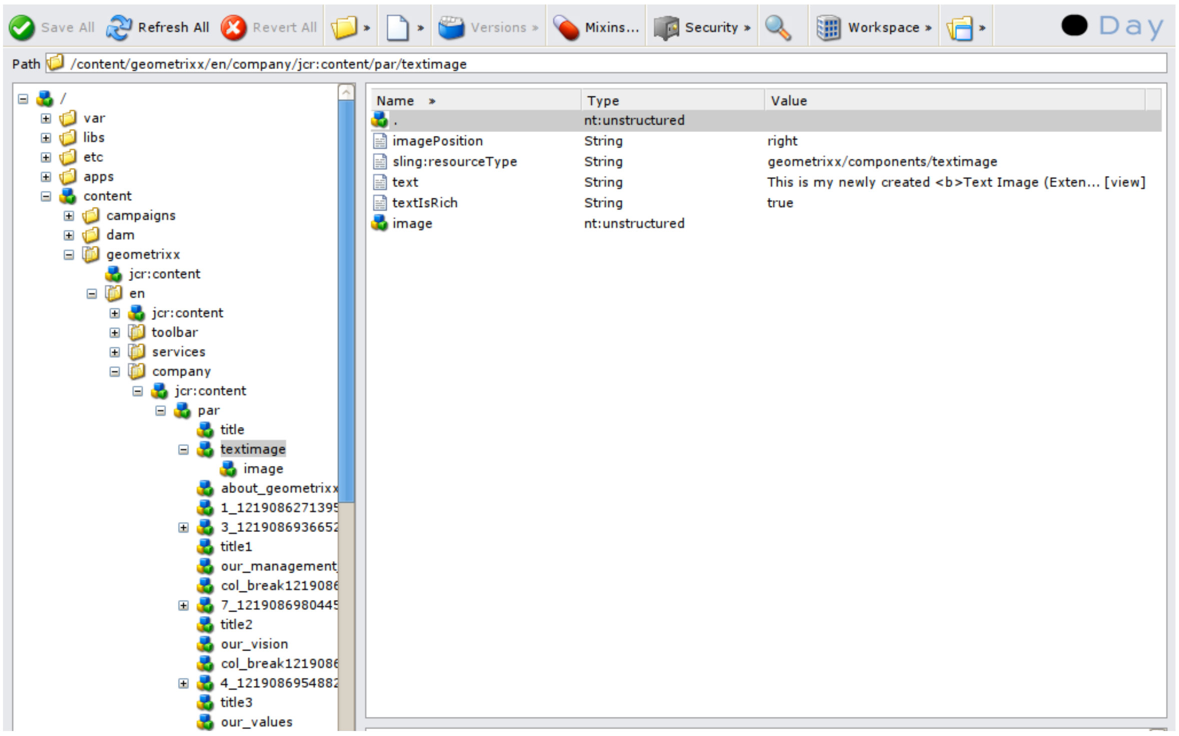
About Geometrixx

This is my newly created **Text Image (Extended)** component!

Geometrixx was founded 2014 libero sem, imperdiet et, fermentum tincidunt, blandit in, libero. Aenean nec nulla non tortor eleifend pellentesque. Pellentesque dui. Duis at eros quis urna pellentesque dapibus. In neque velit, lobortis non, laoreet et, pretium

8. The component is now ready to use.

The component stores its content in a paragraph on the Company page. The following screenshot shows how our new configuration parameter is persisted in the repository, with the node representing the paragraph we have just created.



Path /content/geometrixx/en/company/jcr:content/par/textimage

```
/
  var
  libs
  etc
  apps
  content
    campaigns
    dam
    geometrixx
      jcr:content
      en
        jcr:content
        toolbar
        services
        company
          jcr:content
            par
              title
              textimage
                image
              about_geometrixx
              1_1219086271395
              3_1219086936652
              title1
              our_management
              col_break1219086
              7_1219086980445
              title2
              our_vision
              col_break1219086
              4_1219086954882
              title3
              our_values
```

Name	Type	Value
.	nt:unstructured	
imagePosition	String	right
sling:resourceType	String	geometrixx/components/textimage
text	String	This is my newly created <b>Text Image (Exten... [view]
textIsRich	String	true
image	nt:unstructured	

The `textimage/imagePosition` parameter represents the position of the image for this paragraph on `/content/geometrixx/en/company` page.

7.5 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`.

7.5.1 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.

7.5.1.1 Function of global.jsp, used APIs and Taglibs

The following lists the most important objects provided from the default `global.jsp` (`/libs/wcm/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 CQ content pages (resourceResolver.adaptTo(PageManager.class);).
- component - The component object of the current CQ5 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.

7.5.1.2 Accessing Content

There are three methods to access content in CQ5 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");

Note

Day does not recommend using the JCR API directly in CQ5 if not really necessary; CQ5 is a Sling Application and therefore we deal with resources and not nodes.

7.5.2 JSP Tag Libraries

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

7.5.2.1 CQ Tag Library

The CQ tag library contains helpful CQ 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/wcm/global.jsp` file is included in the script, the cq taglib is automatically declared.

Tip

When you develop the jsp script of a CQ5 component, it is recommended to include following code at the top of the script:

```
<%@include file="/libs/wcm/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.

7.5.2.1.1 <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 attribute:

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

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 this way:

    a. First, if the parameter language of the<cq:setContentBundle> tag is set, this is used.

    b. Otherwise, the locale of the page is looked up. This is taken from the page path (eg. the path `/content/site/en/some/page` produces the "en" locale).

Example:

```
<%@include file="/libs/wcm/global.jsp"%><%
%><%@ page import="com.day.cq.wcm.foundation.forms.ValidationInfo,
                 com.day.cq.wcm.foundation.forms.FormsConstants,
                 com.day.cq.wcm.foundation.forms.FormsHelper,
                 org.apache.sling.api.resource.Resource,
                 org.apache.sling.api.resource.ResourceUtil,
                 org.apache.sling.api.resource.ValueMap" %><%
%><cq:setContentBundle/>
```

7.5.2.1.2 <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>
```

Note

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" %> directive, the file is included at runtime.

Note

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

- When developing CQ5 components, Day 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.

7.5.2.1.3 <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 CQ5 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 CQ 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 CQ5 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/wcm/global.jsp` file is included in the script, the <cq:defineObjects /> tag is automatically included.

7.5.2.1.4 <cq:requestURL>

The <cq:requestURL> tag writes the current request URL to the JspWriter. The two tags <cq:addParam> and <cq:removeParam> 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 occurence 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>
```

7.5.2.1.5 <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>
```

7.5.2.1.6 <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>
```

7.5.2.2 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/wcm/global.jsp` file is included in the script, the sling taglib is automatically declared.

7.5.2.2.1 <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 replaceSelectors="content" />
```

7.5.2.2.2 <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

resourceName

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/>
```

7.6 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.

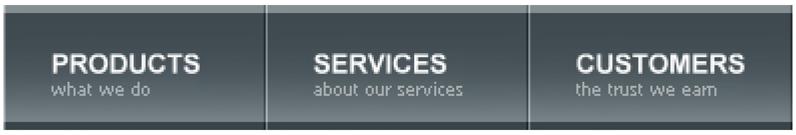
7.6.1 Top Navigation Component

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:



7.6.1.1 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 the parent page on an absolute level
- `PageFilter()`: The default CQ 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 CQ5 WCM.

7.6.1.2 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 CQ5 WCM.

7.6.1.3 Image based Top Navigation Component

The Top Navigation component renders graphical (image based) navigation items.

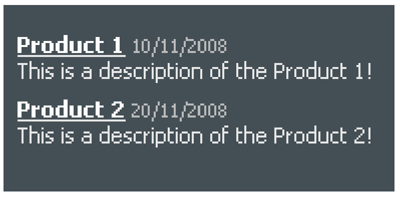
The images for the navigation items are requested from the page resources in a specific view. This means, the paths of the image tags for the navigation items are equivalent to the paths of the pages that the navigation items represent. To identify the view (kind of presentation) the resource (page)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

7.6.2 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:



Product 1 10/11/2008
This is a description of the Product 1!
Product 2 20/11/2008
This is a description of the Product 2!

7.6.2.1 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

7.6.3 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

7.6.3.1 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.

7.6.4 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 CQ5 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.

7.6.5 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.

7.6.5.1 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/*`.

7.6.5.2 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).

7.6.5.3 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 CQ5 WCM.

7.6.5.4 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 section called “Image Rendering Essentials”.

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.

7.6.6 Text & Image Component

The Text & Image Component displays text and images in the main paragraphsystem.

Specification summary:

- `/libs/foundation/components/textimage`
- Displays a text and image in the main paragraphsystem.
- 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)

7.6.6.1 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.

7.6.6.2 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 CQ5 WCM.

7.6.7 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

7.6.7.1 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 CQ5 WCM.

8 Guidelines for Using Templates and Components

CQ components and templates form a very powerful toolkit. They can be used by developers to provide website business users, editors, and administrators with the functionality to adapt their websites to changing business needs (content agility) while retaining the uniform layout of the sites (brand protection).

A typical challenge for a person responsible for a website, or set of websites (for example in a branch office of a global enterprise), is to introduce a new type of content presentation on their websites.

Let us assume there is a need to add a “newslist” page to the websites, which lists extracts from other articles already published. The page should have the same design and structure as the rest of the website.

The recommended way to approach such a challenge would be to:

- Reuse an existing template to create a new type of page. The template roughly defines page structure (navigation elements, panels, and so on), which is further fine-tuned by its design (CSS, graphics).
- Use the paragraph system (parsys/iparsys) on the new pages.
- Define access right to the Design mode of the paragraph systems, so that only authorized people (usually the administrator) can change them.
- Define the components allowed in the given paragraph system so that editors can then place the required components on the page. In our case it could be a “list” component, which can traverse a subtree of pages and extract the information according to predefined rules.
- Editors add and configure the allowed components, on the pages they are responsible for, to deliver the requested functionality (information) to the business.

This illustrates how this approach empowers the contributing users and administrators of the website to respond to business needs quickly, without requiring the involvement of development teams. Alternative methods, such as creating a new template, is usually a costly exercise, requiring a change management process and involvement of the development team. This makes the whole process much longer and costly.

The developers of CQ based systems should therefore use:

- templates and access control to paragraph system design for uniformity and brand protection
- paragraph system including its configuration options for flexibility.

The following general rules for developers make sense in majority of usual projects:

- Keep the number of templates low - as low as the number of fundamentally different page structures on the web sites.
- Provide necessary flexibility and configuration capabilities to your custom components.
- Maximize use of the power and flexibility of CQ paragraph system - the parsys & iparsys components.

9 Java WCM API

The WCM API package, `com.day.cq.wcm.*`, is used to access CQ WCM functionality.

10 Data Modelling

10.1 Data Modeling - David Nuescheler's Model

10.1.1 Source

The following details are ideas and comments expressed by David Nuescheler.

David is co-founder and CTO of Day Software AG, a leading provider of global content management and content infrastructure software. He also leads the development of JSR-170, the Java Content Repository (JCR) application programming interface (API), the technology standard for content management.

Further updates can also be seen on http://wiki.apache.org/jackrabbit/DavidsModel.

10.1.2 Introduction from David

In various discussions I found that developers are somewhat at unease with the features and functionalities presented by JCR when it comes to content modeling. There is no guide and very little experience yet on how to model content in a repository and why one content model is better than the other.

While in the relational world the software industry has a lot of experience on how to model data, we are still at the early stages for the content repository space.

I would like to start filling this void by expressing my personal opinions on how content should be modeled, hoping that this could some day graduate into something more meaningful to the developers community, which is not just "my opinion" but something that is more generally applicable. So consider this my quickly evolving first stab at it.

Important

Disclaimer: These guidelines express my personal, sometimes controversial views. I am looking forward to debate these guidelines and refine them.

10.1.3 Seven Simple Rules

10.1.3.1 Rule #1: Data First, Structure Later. Maybe.

10.1.3.1.1 Explanation

I recommend not to worry about a declared data structure in an ERD sense. Initially.

Learn to love nt:unstructured (& friends) in development.

I think Stefano pretty much sums this one up.

My bottom-line: Structure is expensive and in many cases it is entirely unnecessary to explicitly declare structure to the underlying storage.

There is an implicit contract about structure that your application inherently uses. Let's say I store the modification date of a blog post in a `lastModified` property. My App will automatically know to read the modification date from that same property again, there is really no need to declare that explicitly.

Further data constraints like mandatory or type and value constraints should only be applied where required for data integrity reasons.

10.1.3.1.2 Example

The above example of using a "lastModified" Date property on for example "blog post" node, really does not mean that there is a need for a special nodetype. I would definitely use "nt:unstructured" for my blog post nodes at least initially. Since in my blogging application all I am going to do is to display the lastModified date anyway (possibly "order by" it) I barely care if it is a Date at all. Since I implicitly trust my blog-writing application to put a "date" there anyway, there really is no need to declare the presence of a "lastModified" date in the form a of nodetype.

10.1.3.1.3 Discussion

http://www.nabble.com/DM-Rule-#1:-Data-First,-Structure-Later.-Maybe.-tf4039967.html

10.1.3.2 Rule #2: Drive the content hierarchy, don't let it happen.

10.1.3.2.1 Explanation

The content hierarchy is a very valuable asset. So don't just let it happen, design it. If you don't have a "good", human-readable name for a node, that's probably that you should reconsider. Arbitrary numbers are hardly ever a "good name".

While it may be extremely easy to quickly put an existing relational model into a hierarchical model, one should put some thought in that process.

In my experience if one thinks of access control and containment usually good drivers for the content hierarchy. Think of it as if it was your file system. Maybe even use files and folders to model it on your local disk.

Personally I prefer hierarchy conventions over the nodetyping system in a lot of cases initially, and introduce the typing later.

10.1.3.2.2 Example

I would model a simple blogging system as follows. Please note that initially I don't even care about the respective nodetypes that I use at this point.

```
/content/myblog
/content/myblog/posts
/content/myblog/posts/what_i_learned_today
/content/myblog/posts/iphone_shipping

/content/myblog/comments/iphone_shipping/i_like_it_too
/content/myblog/comments/iphone_shipping/i_like_it_too/i_hate_it
```

I think one of the things that become apparent is that we all understand the structure of the content based on the example without any further explanations.

What may be unexpected initially is why I wouldn't store the "comments" with the "post", which is due to access control which I would like to be applied in a reasonably hierarchical way.

Using the above content model I can easily allow the "anonymous" user to "create" comments, but keep the anonymous user on a read-only basis for the rest of the workspace.

10.1.3.2.3 Discussion

http://www.nabble.com/DM-Rule-#2:-Drive-the-content-hierarchy,-don't-let-it-happen.-tf4039994.html

10.1.3.3 Rule #3: Workspaces are for clone(), merge() and update().

10.1.3.3.1 Explanation

If you don't use clone(), merge() or update() methods in your application a single workspace is probably the way to go.

"Corresponding nodes" is a concept defined in the JCR spec. Essentially, it boils down to nodes that represent the same content, in different so-called workspaces.

JCR introduces the very abstract concept of Workspaces which leaves a lot of developers unclear on what to do with them. I would like to propose to put your use of workspaces to the following to test.

If you have a considerable overlap of "corresponding" nodes (essentially the nodes with the same UUID) in multiple workspaces you probably put workspaces to good use.

If there is no overlap of nodes with the same UUID you are probably abusing workspaces.

Workspaces should not be used for access control. Visibility of content for a particular group of users is not a good argument to separate things into different workspaces. JCR features "Access Control" in the content repository to provide for that.

Workspaces are the boundary for references and query.

10.1.3.3.2 Example

Use workspaces for things like:

- v1.2 of your project vs. a v1.3 of your project
- a "development", "QA" and a "published" state of content

Do not use workspaces for things like:

- user home directories
- distinct content for different target audiences like public, private, local, ...
- mail-inboxes for different users

10.1.3.3.3 Discussion

http://www.nabble.com/DM-Rule-#3:-Workspaces-are-for-corresponding-nodes.-tf4040010.html

10.1.3.4 Rule #4: Beware of Same Name Siblings.

10.1.3.4.1 Explanation

While Same Name Siblings (SNS) have been introduced into the spec to allow compatibility with data structures that are designed for and expressed through XML and therefore are extremely valuable to JCR, SNS come with a substantial overhead and complexity for the repository.

Any path into the content repository that contains an SNS in one of its path segments becomes much less stable, if an SNS is removed or reordered, it has an impact on the paths of all the other SNS and their children.

For import of XML or interaction with existing XML SNS maybe necessary and useful but I have never used SNS, and never will in my "green field" data models.

10.1.3.4.2 Example

Use

```
/content/myblog/posts/what_i_learned_today
/content/myblog/posts/iphone_shipping
```

instead of

```
/content/blog[1]/post[1]
/content/blog[1]/post[2]
```

10.1.3.4.3 Discussion

http://www.nabble.com/DM-Rule-#4:-Beware-of-Same-Name-Siblings.-tf4040024.html

10.1.3.5 Rule #5: References considered harmful.

10.1.3.5.1 Explanation

References imply referential integrity. I find it important to understand that references do not just add additional cost for the repository managing the referential integrity, but they also are costly from a content flexibility perspective.

Personally I make sure I only ever use references when I really cannot deal with a dangling reference and otherwise use a path, a name or a string UUID to refer to another node.

10.1.3.5.2 Example

Let's assume I allow "references" from a document (a) to another document (b). If I model this relation using reference properties this means that the two documents are linked on a repository level. I cannot export/import document (a) individually, since the reference property's target may not exist. Other operations like merge, update, restore or clone are affected as well.

So I would either model those references as "weak-references" (in JCR v1.0 his essentially boils down to string properties that contain the uuid of the target node) or simply use a path. Sometimes the path is more meaningful to begin with.

I think there are use cases where a system really can't work if a reference is dangling, but I just can't come up with a good "real" yet simple example from my direct experience.

10.1.3.5.3 Discussion

http://www.nabble.com/DM-Rule-#5:-References-considered-harmful.-tf4040042.html

10.1.3.6 Rule #6: Files are Files are Files.

10.1.3.6.1 Explanation

If a content model exposes something that even remotely *smells* like a file or a folder I try to use (or extend from) nt:file, nt:folder and nt:resource.

In my experience a lot of generic applications allow interaction with nt:folder and nt:files implicitly and know how to handle and display those event if they are enriched with additional meta-information. For example a direct interaction with file server implementations like CIFS or WebDAV sitting on top of JCR become implicit.

I think as good rule of thumb one could use the following: If you need to store the filename and the mime-type then nt:file/nt:resource is a very good match. If you could have multiple "files" an nt:folder is a good place to store them.

If you need to add meta information for your resource, let's say an "author" or a "description" property, extend nt:resource not the nt:file. I rarely extend nt:file and frequently extend nt:resource.

10.1.3.6.2 Example

Let's assume that someone would like to upload an image to a blog entry at:

```
/content/myblog/posts/iphone_shipping
```

and maybe the initial gut reaction would be to add a binary property containing the picture.

While there certainly are good use cases to use just a binary property (let's say the name is irrelevant and the mime-type is implicit) in this case I would recommend the following structure for my blog example.

```
/content/myblog/posts/iphone_shipping/attachments [nt:folder]
/content/myblog/posts/iphone_shipping/attachments/front.jpg [nt:file]
/content/myblog/posts/iphone_shipping/attachments/front.jpg/jcr:content [nt:resource]
```

10.1.3.6.3 Discussion

http://www.nabble.com/DM-Rule-#6:-Files-are-Files-are-Files.-tf4040063.html

10.1.3.7 Rule #7: IDs are evil.

10.1.3.7.1 Explanation

In relational databases IDs are a necessary means to express relations, so people tend to use them in content models as well. Mostly for the wrong reasons through.

If your content model is full of properties that end in "Id" you probably are not leveraging the hierarchy properly.

It is true that some nodes need a stable identification throughout their live cycle. Much fewer than you might think though. mix:referenceable provides such a mechanism built into the repository, so there really is no need to come up with an additional means of identifying a node in a stable fashion.

Keep also in mind that items can be identified by path, and as much as "symlinks" make way more sense for most users than hardlinks in a unix filesystem, a path makes a sense for most applications to refer to a target node.

More importantly, it is **mix**:referenceable which means that it can be applied to a node at the point in time when you actually need to reference it.

So let's say just because you would like to be able to potentially reference a node of type "Document" does not mean that your "Document" nodetype has to extend from mix:referenceable in a static fashion since it can be added to any instance of the "Document" dynamically.

10.1.3.7.2 Example

use:

```
/content/myblog/posts/iphone_shipping/attachments/front.jpg
```

instead of:

```
[Blog]
- blogId
- author

[Post]
- postId
- blogId
- title
- text
- date

[Attachment]
- attachmentId
- postId
- filename
```

```
+ resource (nt:resource)
```

10.1.3.7.3 Discussion

http://www.nabble.com/DM-Rule-#7:-IDs-are-evil.-tf4040076.html

Appendix A. Keyboard Shortcuts

Table A.1. Keyboard Shortcuts

Location	Shortcut	Description
Page with a teaser component.	**`Ctrl-Alt-c`**	Shows the clickstream cloud: collection of page tags related to clicks that the user has made and used to focus the teaser.
Content Finder - Search box	**`down-arrow`**	Trigger a suggestions list. Needed when too few characters have been entered to trigger the list automatically (this happens when 2 or more characters have been entered).
	`right-arrow` (on a suggested path)	Select item and trigger suggestions for the selected path.
	`left-arrow` (on a suggested path)	Select item and trigger suggestions for its ancestors (as in siblings of parent).
	`Enter` (on a suggested path)	Select item and trigger search.
	`Esc`	Close suggestions layer.
Drag assets, drop on destination	**`Alt+drag`**	The drop action will produce a new paragraph; instead of replacing the asset in the destination.
Content Window (Edit Mode) - Paragraphs	**`Shift-Click`**	Select multiple paragraphs.
	`Ctrl-Click`	Select multiple paragraphs.
	`Ctrl-C`	Copy selected paragraph(s).
	`Ctrl-X`	Cut selected paragraph(s). **Note:** *The cut paragraph will not disappear until it has been pasted to the new location.*
	`Ctrl-V`	Paste paragraphs from clipboard.
	`Alt-Ctrl-V`	Paste as reference.
	`Del`	Delete selected paragraph(s).
	`Backspace`	Delete selected paragraph(s).
	`Alt-right-click`	Force default (brower) context menu.

Table A.2. Extended Keyboard Shortcuts

Location	Shortcut	Description
Page	**`Ctrl-Shift-U`** with `?debugClientLibs=true` set in the URL.	To see timing statistics for page loading.

Appendix B. Security Checklist

B.1 Security Checklist for Developers

B.1.1 Use the user session, not the administrative session

This means you should use:

```
slingRequest.getResourceResolver().adaptTo(Session.class);
```

Appendix C. Copyright, Licenses and Formatting Conventions

For all copyright statements and license agreements see Copyright, Licenses and Disclaimers.

C.1 Formatting Conventions

The following tables detail formatting conventions used within this guide:

Table C.1. Formatting Conventions - Text

Style	Description	Example
Cross-reference	Cross-reference to external documents.	See the *Microsoft Manual of Style for Technical Publications*.
`GUI Item`	User interface items.	Click **`Save`**.
`Keyboard shortcut`	Keyboard shortcuts.	Press **`Ctrl+A`**.
`Mouse Button`	Mouse buttons.	**`Secondary-mouse`** button (usually the **`right-mouse`** button).
Link	Link to anchor-points within the current document and/or external sources.	http://www.day.com
`Code`	Example of programming code.	`if (weather == sunny) smile;`
`User Input`	Example of text, or commands, that you type.	**`ls *.xml`**
`<Variable User Input>`	Example of variable text - you type the actual value needed.	`ls <cq-installation-dir>`
`[Optional Parameter]`	An optional parameter.	`ls [<option>] [<filename>]`
`Computer Output`	Logging and error messages.	`ls: cannot access error.log:`

Table C.2. Formatting Conventions - Actions

When you see this...	It means do this...
`Ctrl+A`	Hold down the **`Ctrl`** key, then press the **`A`** key.
`Right-click`	Press the **`right-mouse`** button (or the **`left-mouse`** button if your mouse has been configured for left-handed use).
`Drag`	Hold down the left mouse button while moving the item, then release the mouse button at the new location (or the right mouse button if your mouse has been configured for left-handed use).