

CQ5 Connector Guide

**VIA Best Practice Office**

<http://www.vamosa.com>

Document Control

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
|  | A Mancevice | 1.0 |  |
|  |  |  |  |
|  |  |  |  |

* 1. Reviewers

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Company | Job Title and/or Project Role | Date Reviewed |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* 1. Approvers

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Company | Job Title and/or Project Role | Date Approved |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 4](#_Toc253663262)

[1.1 Document Purpose 4](#_Toc253663263)

[1.2 Intended Audience 4](#_Toc253663264)

[1.3 Accessing CQ content explorer (CRX) 4](#_Toc253663265)

[2 Deploying the connector in the target environment 5](#_Toc253663266)

[2.1 Deploying the connector 5](#_Toc253663267)

[2.2 Instantiating the connector 6](#_Toc253663268)

[2.3 Setting Up WebDAV 6](#_Toc253663269)

[3 The VCM-side loader 7](#_Toc253663270)

[3.1 Loading Assets 7](#_Toc253663271)

[3.2 Loading Pages 7](#_Toc253663272)

[3.3 Loading IA 7](#_Toc253663273)

[4 Creating Content 8](#_Toc253663274)

[4.1 How the connector works 8](#_Toc253663275)

[4.2 Creating an asset 8](#_Toc253663276)

[4.2.1 The <asset> element 8](#_Toc253663277)

[4.2.2 The <node> element 8](#_Toc253663278)

[4.2.3 The <prop> element 8](#_Toc253663279)

[4.2.4 Example payload 10](#_Toc253663280)

[4.3 Creating a page 11](#_Toc253663281)

[4.3.1 The <page> element 11](#_Toc253663282)

[4.3.2 The <node> element 11](#_Toc253663283)

[4.3.3 The <prop> element 11](#_Toc253663284)

[4.3.4 Example payload 13](#_Toc253663285)

[5 Additional Resources 14](#_Toc253663286)

[5.1 Example Skeletons 14](#_Toc253663287)

[5.2 Example IA Spreadsheet 14](#_Toc253663288)

[5.3 Example Query 14](#_Toc253663289)

Terms & Conditions

Copyright © Vamosa Ltd

All rights reserved, no part of this document may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of Vamosa.

This document is for the use of the recipient(s) only. None of its content may be disclosed to third parties without the prior written permission of Vamosa. Such disclosure will only be made on the basis that the contents are not passed to persons other than those specified in the consent. In addition, neither the whole nor any part of this document, nor any reference thereto may be included in any document, circular or statement without the prior written approval by Vamosa of the form and context in which it will appear.

Introduction

Document Purpose

This document will explain how to deploy and use the connector files for migrating into Day CQ5 CMS. These procedures must be completed prior to the development phase of a migration and are required to ensure a path from source to target environments is established and therefore facilitate a successful end to end migration. As a consequence the execution of this test pack is a key indicator to the project state and readiness to migrate.

Intended Audience

This document assumes the user has experience using:

* VCM Content Migrator
* XML
* Code development platforms such as Eclipse
* Javascript
* Java

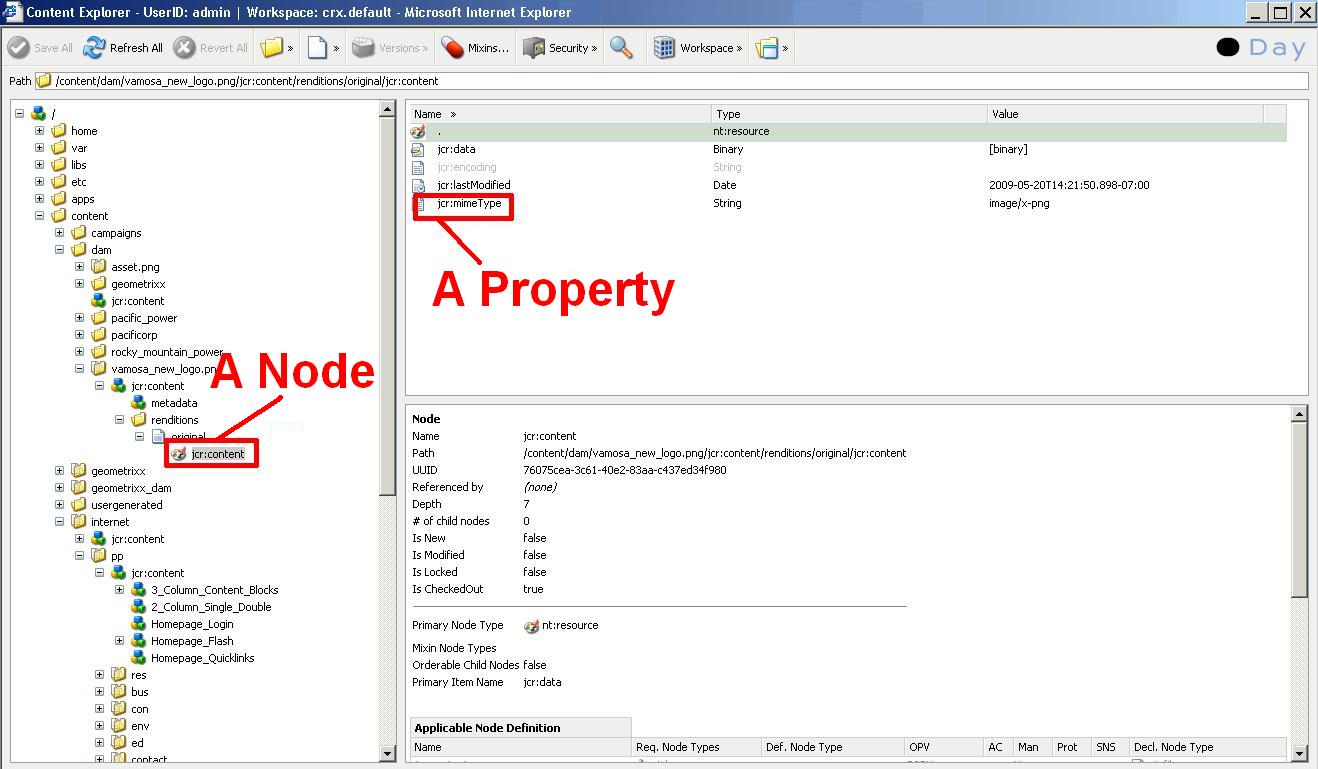
And, while not required, familiarity with the CQ5 documentation is strongly encouraged.

Accessing CQ content explorer (CRX)

CQ comes with a content explorer that can be very useful in both familiarizing one’s self with CQ5 and verifying page/node/property values. To access, log into CQ5 through Internet Explorer. Append ‘/crx’ to URL and log into CRX (admin\admin is a common login\password). The content explorer link will now bring the user to a view of the content repository.

This tool is essential in understanding how content is populated. If unclear on how to create a specific object, creating said object using the CQ GUI and viewing it within CRX will give the user a much clearer understanding of what is required of the XML payload to instantiate aspects of the content, be it a page or an asset.

Note that in CQ5, all objects can be thought of as either a “Node” or a “Property” of a node. This can be confusing since a node may have one of many possible types (which are represented by a collection of different icons). An easy way to differentiate between the two is that a node is always viewable in the left-hand panel, while both properties and nodes are viewable in the center panel.

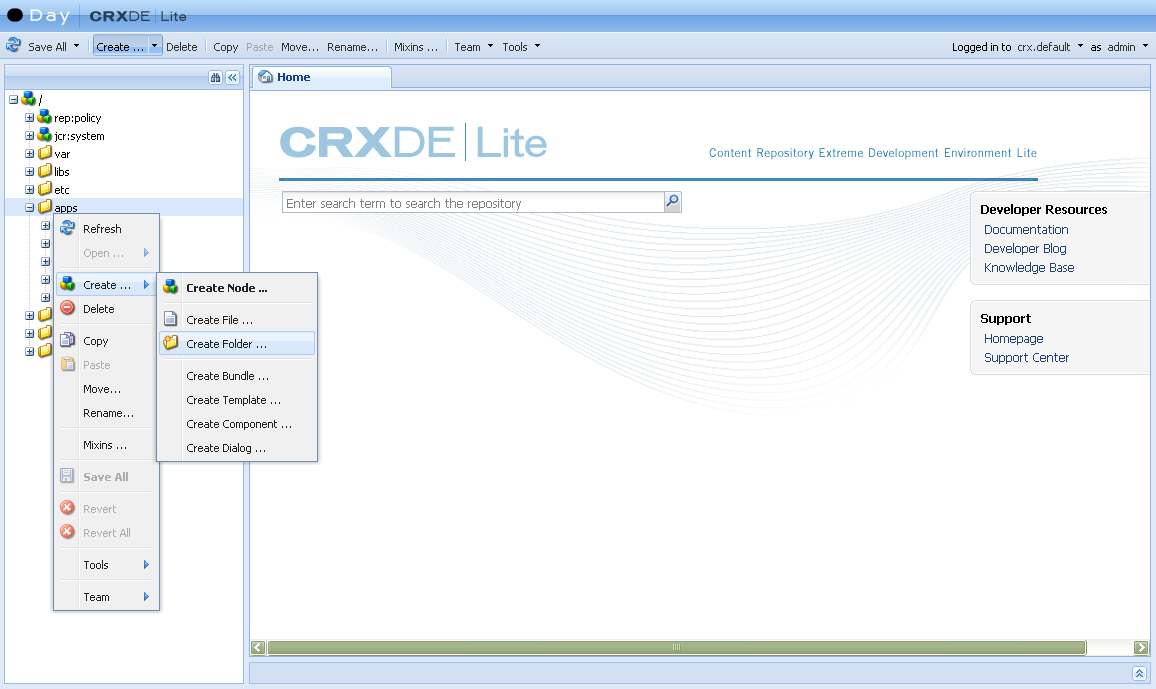


Deploying the connector in the target environment

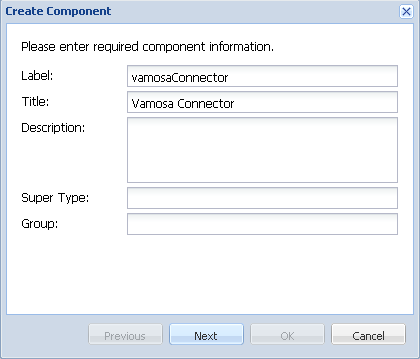
Deploying the connector

To deploy the connector the .jsp file must be dropped into the CQ environment (NOTE: It is highly advisable to select “Save All” after each step)

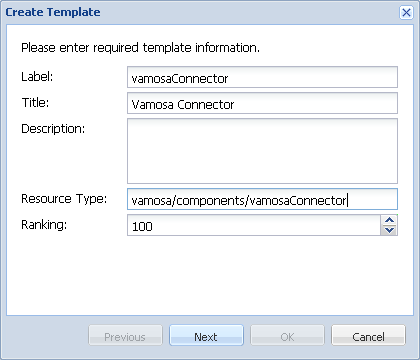
* Navigate to the CRX explorer (append /crx to the CQ5 domain) and log in
* Click the link to “CRXDE Lite”
* It may be necessary to log into CRXDE as well; log in via the top-right corner
* Using the left-hand tree navigation, browse to the /apps directory and create a new folder “vamosa”. Then create the directories “components” and “directories” beneath this folder.



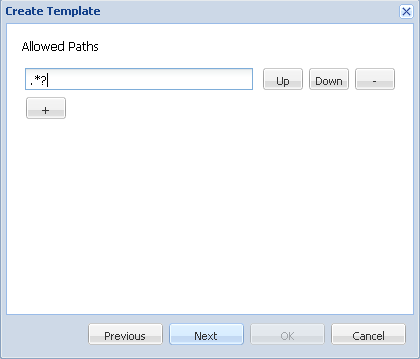
* Right-click on the components directory and select Create🡪Create Component
* Enter the values “vamosaConnector” and “Vamosa Connector” in the label/title fields of the dialog. Click through the rest of the dialog



* Navigate to the created vamosaConnector.jsp and paste in the connector code
* Right-click on the templates directory and select Create🡪Create Template
* Enter the same values into the label/title fields as before. In the Resource Type field enter “vamosa/components/vamosaConnector”



* Click “Next” and add the allowedPath value “.\*?”



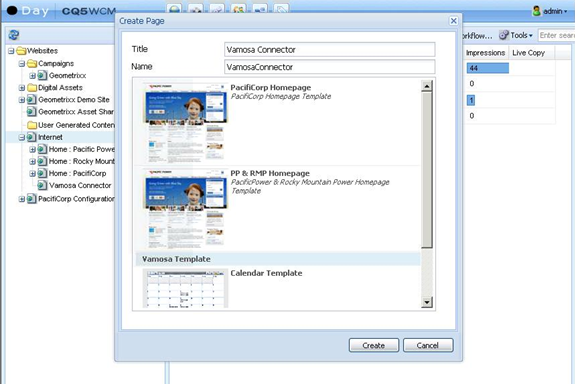
* Click through the remainder of the dialog.
* Click the “Save All” button in the top-left of CRXDE

The template is now ready to be used in creating the connector page.

Instantiating the connector

To use the connector, a page must be created using the connector template.

* Open CQ
* Navigate to area for content publication
* Create new page, selecting the new template as the page template



* Open page and insert xml payload into form
* Submit
* Ensure a page has been created

Setting Up WebDAV

Setting up WebDAV is as simple as installing the NetDrive software included in the connector package and mapping a drive to the CQ5 instance. To Map a drive:

* Open NetDrive
* Select “New Site”
* Give the site a name
* Paste the URL of the CQ5 instance in the “Site IP or URL” field
* Enter the port number into the “Port” field (if given in the URL for CQ5)
* Set Server Type to “WebDAV” and Pick a drive letter
* Enter login details
* Connect!

The VCM-side loader

The CQ5 Loader can be used to load assets, pages, or placeholders into the CQ5 system. Assets are loaded using the per-object “enhance” method while pages and placeholders are loaded on a per-project basis. All three types require parameters specifying the CQ login credentials and connector URL but don’t require inputs for methods that are not used.

Loading Assets

Loading an asset requires an XML skeleton into which both the raw Base-64 data and any metadata may be inserted. The loader generates a payload on the fly using the XML skeleton provided as a parameter. In addition, an asset requires at minimum four (4) data fields to be loaded:

* Path – an absolute path to the asset in the file system
* Filename – the filename of the asset
* Last Modified Date – date of most recent modification
* Data – the Base64-encoded string representing the objects’ binary data

The first three fields are pulled from the asset’s metadata attributes; the names of which are specified by the “asset\_”-prefixed input parameters. The data field is pulled from the VCM-stored content data and does not require any modification.

Any additional metadata properties of the asset should be written into the XML skeleton under the “metadata” node. As mentioned earlier, these properties *must* be prefixed in order to for CQ to properly instantiate the asset. Failing to do so can result in corrupting the containing folder (a retroactive fix using the CRX tool is easily done, however). Usually this prefix is “dam” or “dc” the latter for titles and the former for metadata. Experimenting using the GUI and CRX tools concurrently will reinforce the concept.

It should be noted that any dates—for assets and pages alike—pulled from metadata must be of the form “yyyy-MM-dd'T'hh:mm:ss.SSSZ” (eg. 1992-09-11T10:25:52.073-05:00) as specified by the Java [SimpleDateFormat](http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html) Object. However, a function has been provided in the CQ5 connector package to convert a given date into this format.

Loading Pages

Pages are loaded on a per-project basis. This is done so that pages may be loaded in an order that avoids sending payloads to non-existent IA levels. Loading pages requires a query library parameter and the string specifying the query to use to be entered as parameters of the loader. Assuming the pages have been cut up into the appropriate XML format and the query returns pages in descending order of architecture, no other inputs are required.

Loading IA

The CQ5 loader also has the ability to load placeholder pages based on an Excel spreadsheet. For instance, if the content architecture is changing in the migration and source pages are being moved underneath target pages that do not currently exist in the source site. The Excel resource requires at least four columns named:

* “Path” – absolute path to the page
* “Filename” – name of the page in the file system
* “Template Path” – absolute path to the template for the page
* “Label” – the title of the page

Running this method before the load ensures that pages’ paths exist before the content load.

Creating Content

How the connector works

The connector is essentially a unique type of page within the CQ environment. For testing purposes, when viewed, the connector appears as an HTML form. Submitting a payload to the form adds a CQ property, “payload,” to the page with the submitted XML text as the property’s value and automatically calls a refresh of the page. Upon refresh, the connector skips over the form and begins parsing the XML payload and executing commands. Only one asset or page object may be submitted at a time.

Creating an asset

Assets are most easily created by gaining access to the WebDAV directory and uploading files to this directory. However assets can be created using an XML payload as well, or even have additional metadata included with the WebDAV protocol. This section describes creating an asset from an XML payload. (Note, this is generally not how assets should be loaded, but rather to be used to “paint” metadata over created assets)

The <asset> element

An **<asset>** element is the topmost element in an asset’s creation. An <**asset>** may have only children of type **<node>**. Assets require three (3) inputs for creation, which are stored as attributes of the **<asset>** element

* filename: The filename of the asset
* path: The intended path to the asset
* type: The node-type of the asset (usually “dam:Asset”)

A path for an asset may be set to a nonexistent location. The method used to create the asset object will create intermediary nodes with the type “sling:Folder”. This is permitted since assets do not have a parent-child relationship with other assets (like pages).

The <node> element

Because assets are not created using any sort of template the node payload must be very explicit. To ensure proper payloads are created it is recommended that GUI-instantiated assets’ internal nodes & properties are observed within the CRX repository so they may be replicated in the XML payload.

Nodes require two (2) attributes for creation but require additional **<prop>** elements to function properly. The attributes needed to create a node are:

* path: The path of the node relative to the page
* type: The type of node. For assets this type is very meaningful

All other nodes must be explicitly declared and not implied through children (ie,a node jcr:content/aNode but be declared before jcr:content/aNode/anotherNode), even if said node has no properties.

The <prop> element

Properties of components (nodes) are set using **<prop>** elements. The property of the component can define the behaviour of or write data to said component. Properties require the following attributes:

* name: The name of the property. Properties for assets must have a prefix.
* type: The data type for the value of the property. Acceptable types are:
  + String - any text (HTML characters must be escaped)
  + Binary - Text in Base64
  + Long - long data type value (eg. 11091992)
  + Double - double data type (eg. 11.09)
  + Date - text of form: ±YYYY-MM-DDThh:mm:ss.SSSTZD
  + Boolean - true/false
  + Path – Unknown
  + Name – Unknown
  + Reference - UUID of referenceable node (format not known)

In addition, any data type may be suffixed with “[]” to indicate and array of said type.

The value of a **<prop>** element is set as the property value. Ensure that the value is typed properly. If the data is an array, the values are comma-separated.

For assets, the “jcr:data” property of the “jcr:content” node contains the data representing the asset, which is contained as a Base64 encoded string. Note that the properties “jcr:data” and “jcr:lastModified” are mandatory for. The connector will fail to instantiate the asset if a mandatory property is not explicitly created. Additionally, all assets’ metadata is stored in a node titled “metadata” with the relative path “/jcr:content/metadata”. These properties must be prefixed or else the asset’s containing folder will cease to display any assets at all. Certain prefixes have different functionality and should be explored when the asset skeleton is created. Properly instantiated properties will ensure that the GUI-side of CQ will function as expected.

Example payload

<asset filename="vamosa.png" path="/content/dam" type="dam:Asset">

<node path="jcr:content" type="dam:AssetContent"/>

<prop name="cq:tags" type="String[]">vtag1,vtag2,vtag3</prop>

<node path="jcr:content/metadata" type="nt:unstructured">

<prop name="dc:description" type="String">desc</prop>

<prop name="dc:format" type="String">image/x-png</prop>

<prop name="dc:title" type="String">Vamosa</prop>

<prop name=”dam:myMetaField” type=”String”>metadata</prop>

</node>

<node path="jcr:content/renditions" type="nt:folder"/>

<node path="jcr:content/renditions/original" type="nt:file"/>

<node path="jcr:content/renditions/original/jcr:content" type="nt:resource">

<prop name="jcr:lastModified" type="Date">2009-02-10T07:25:52.073-08:00</prop>

<prop name="jcr:data" type="Binary"> </prop>

<prop name="jcr:mimeType" type="String">image/x-png</prop>

</node>

</asset>

Creating a page

This section describes creating a page from an XML payload.

The <page> element

A **<page>** element is the topmost element in a page’s creation. A **<page>** may have only children of type **<node>**. Pages require four (4) inputs for creation, which are stored as attributes of the **<page>** element:

* path: The path of the page in the target environment
* filename: The name of the page in the file system
* template\_path: The path to the target template for the page
* label: The display title of the page

This is all that is required to create a page. It should be noted, however that, unlike assets, a page can only be created if the path to it exists. In other words the page “content/aPage” must exist before creating “content/aPage/bPage”

It should be noted that because templates can have an “allowedPath” attribute, pages created using a template with a restricted path will require a “path” attribute that conforms to the template’s allowed path. This enables CQ to disallow the creation of certain pages above their specified IA level. In addition the “filename” attribute must contain only standard filename characters (no spaces, &c.).

The <node> element

Creating a page instantiates the standard “jcr:content” node. In addition, the template may instantiate other nodes common to the template. These nodes may be omitted from the XML payload (though including them would not do any harm—providing that no aspects of the node specified are altered in the instance). All other nodes, however, must be explicitly declared and not implied through children (ie,a node jcr:content/aNode but be declared before jcr:content/aNode/anotherNode), even if said node has no properties.

Nodes are components on the page. A page can be created with any number of nodes, including none. Nodes may only have **<page>** elements as parents and **<prop>** elements as children. They require two (2) attributes for creation but require additional **<prop>** elements to function properly. The attributes needed to create a node are:

* path: The path of the node relative to the page
* type: The type of node. Almost always this type is “nt:unstructured” for pages

A full list of node types can be viewed by creating a new node in the CRX repository and viewing the drop-down menu under “type.” Note that this

The <prop> element

Properties of components (nodes) are set using **<prop>** elements. The property of the component can define the behaviour of or write data to said component. Properties require the following attributes:

* name: The name of the property. This is usually significant and can define the role of the property in the component based on the type of component
* type: The data type for the value of the property. Acceptable types are:
  + String - any text (HTML characters must be escaped)
  + Binary - Text in Base64
  + Long - long data type value (eg. 11091992)
  + Double - double data type (eg. 11.09)
  + Date - text of form: ±YYYY-MM-DDThh:mm:ss.SSSTZD
  + Boolean - true/false
  + Path – Unknown
  + Name – Unknown
  + Reference - UUID of referenceable node (format not known)

In addition, any data type may be suffixed with “[]” to indicate and array of said type.

The value of a **<prop>** element is set as the property value. Ensure that the value is typed properly. If the data is an array, the values are comma-separated.

**IMPORTANT NOTE:** Since all **<node>** elements represent components, they must contain a **<prop>** defining the type of component—like a component template. These components can be created by the client or exist out-of-the-box from Day. This **<prop>** element will have the name “sling:resourceType” the type “String” and the value will be the path to the intended component.

Example payload

<page path="/content/geometrixx/" filename="Vamosa" template\_path="/apps/geometrixx/templates/contentpage" label="Vamosa">

<node path="jcr:content/Linked\_Image" type="nt:unstructured">

<prop name="description" type="String">An image description</prop>

<prop name="fileReference" type="String">/content/dam/asset.png</prop>

<prop name="imageCaption" type="String">&lt;an image caption&gt;</prop>

<prop name="linkURL" type="String"></prop>

<prop name="sling:resourceType" type="String">pc\_global/components/linkedimage</prop>

</node>

<node path="jcr:content/General\_Content" type="nt:unstructured">

<prop name="sling:resourceType" type="String">pc\_global/components/pcparsys</prop>

</node>

<node path="jcr:content/General\_Content/pcreference" type="nt:unstructured">

<prop name="sling:resourceType" type="String">pc\_global/components/pcreference</prop>

<prop name="path" type="String"> /content/internet/pp/vamosa\_test/jcr:content/General\_Content/pcbasiccontentblock

</prop>

</node>

<node path="jcr:content/General\_Content/text" type="nt:unstructured">

<prop name="sling:resourceType" type="String">foundation/components/text</prop>

<prop name="text" type="String">&lt;b&gt;Welcome To Vamosa's test page!&lt;/b&gt;</prop>

<prop name="textIsRich" type="Boolean">true</prop>

</node>

<node path="jcr:content/General\_Content/chart" type="nt:unstructured">

<prop name="sling:resourceType" type="String">foundation/components/chart</prop>

<prop name="chartAlt" type="String">Vamosa Alt Text</prop>

<prop name="chartHeight" type="String">400</prop>

<prop name="chartWidth" type="String">400</prop>

<prop name="chartType" type="String">bar</prop>

</node>

</page>

Additional Resources

Example Skeletons

Two example XML skeletons have been provided to help get started in creating custom project-specific templates. Notice the attributes prefixed with the “c:” namespace and how they are used in the cutup script.

Example IA Spreadsheet

An example spreadsheet is provided to display the format necessary for the IA-Load method in the VCM CQ Loader. Note that the data is organized hierarchically so that pages are loaded before any existing children.

Example Query

Finally, an example query is provided to show how to return pages in order during the final content load. To reiterate, content must be loaded in order to avoid attempting to create pages in nonexistent IA paths.