Day

Technical Training

Exercise Guide

CQ 5.3 Developer Training

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230

CQ 5.3 Developer Training

V 1.6

Published December 2010
Copyright © 2010 Day Management AG, Switzerland. All rights reserved.

No part of this publication (hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher. Information in this manual is furnished under license by Day Software, Inc. and may only be used in accordance with the terms of the software license.

Day, the Day logo, Communiqué and CRX are registered trademarks and service marks, or are trademarks and service marks of Day Management AG, Switzerland, in various countries around the world. All other product names and company logos mentioned in the information, documents or other items provided or available herein may be the trademarks of their respective owners.

We grant you limited license to use materials solely for eduction purposes. You acknowledge that all right, title and interest (including IP rights) in and to the materials remains in Day.

Day 1 6

EXERCISE - Install & Start an Author Instance 6

EXERCISE - Browse Related Application/Server Interfaces 9

EXERCISE - Create an Application/Project 14

EXERCISE - Create a Template 19

EXERCISE - Create a "Page" Component 23

EXERCISE - Create Pages & Web Site Structure 26

EXERCISE - Install & Start CRXDE 30

EXERCISE - Utilize CRXDE 32

EXERCISE - Include the "global.jsp" in the Page Component 36

EXERCISE - Display Basic Page Content 39

Day 2 42

EXERCISE - Create Multiple Scripts/Renderers for the "Page" Component 42

EXERCISE - Breakout/Modularize the "Page" Component 50

EXERCISE - Initialize the WCM 53

EXERCISE - Extend the Foundation "Page" Component 55

EXERCISE - Extend the Script Structure of the "Page" Component 60

EXERCISE - Create and Assign a New Design 64

EXERCISE - Create a Text-based Navigation Component 71

EXERCISE - Add a Log Message 76

EXERCISE - Enable the Debugger 80

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create an Image-based Navigation Component 85

EXERCISE - Create a Title Component 89

Extra Credit EXERCISE - Create a List Children Component 96

Day 3 98

EXERCISE - Create a Logo Component 98

EXERCISE - Include the Foundation Breadcrumb Component 107

Extra Credit EXERCISE - Modify the Foundation Breadcrumb component 110

Extra Credit EXERCISE - Modify your logo component 111

Extra Credit EXERCISE - Modify your topnav component 112

EXERCISE - Include the Foundation Paragraph System Component 113

EXERCISE - Content Finder Drag-and-Drop 118

EXERCISE - Create a Complex Component 123

EXERCISE - Include Multiple Foundation Components 136

EXERCISE - Create a Search Component 139

EXERCISE - Apply i18n to a Component 145

EXERCISE - Create & Register a Widget 150

Day 4 158

EXERCISE - Create and Consume an OSGi Bundle 158

EXERCISE - Examine the Workflow Console 167

EXERCISE - Create a Workflow Implementation Step 174

EXERCISE - Create & Download a CQ Package 182

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Find Slow Responses 188

EXERCISE - Change Default Passwords 199

Appendix A 210

Exercise - Upgrading CQ 5.3 to a CRX 2.1 Repository 210

Appendix B 216

Exercise - Clone an Author Instance to be a Publish Instance 216

Copyright 2010, Day Software AG, Switzerland

Day 1
EXERCISE - Install & Start an Author Instance

Goal

The following instructions explain how to install and start an Author instance. This is important because you will use this Author instance throughout this training to perform typical development tasks. To successfully complete and understand these instructions, you will need:

- A CQ5 quickstart JAR
- A valid CQ5 license key
- A JDK >= 1.5
- Approximately 800 MBs of free space
- Approximately 1 GB of RAM

What is an Author instance?

An Author instance is the CQ5 installation content authors will login to and manage pages. This includes: 1) creating, 2) editing, 3) deleting, 4) moving, 5) etc. In addition, it is the installation you will be developing against as you can easily observe both Author and Publish views.

How to install an Author instance:

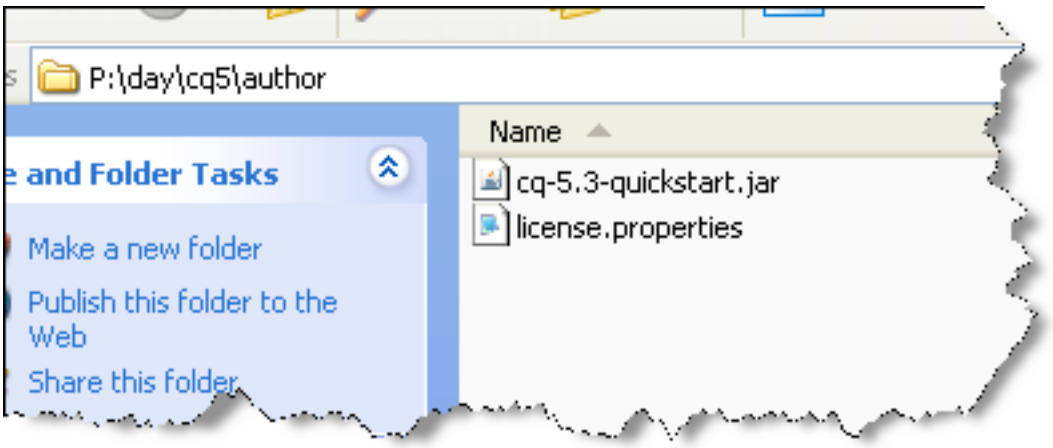
1. Create a folder structure on your file system where you will store, install, and start CQ5 (e.g. C:/day/cq5/author).

WARNING

MS Windows users, please do not use spaces in your newly created folder structure (e.g. C:/this is bad/cq5/author). This will cause CQ5 to error.

2. Copy the CQ5 quickstart JAR and license.properties file from <USB>/distribution/cq5_wcm into the newly created folder structure.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



CQ5 author install folder structure

3. Rename the CQ5 quickstart JAR to *cq-author-4502.jar*.

- cq = the application
- author = the WCM mode it will run in (e.g. author or publish)
- 4502 = the port it will run in (e.g. any available port is acceptable)

NOTE

If no port number is provided in the file name, CQ5 will select the first available port from the following list: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, 7) 8085, 8) 8888, 9) 9362, 10) random.

4. Double-click the cq-author-4502.jar file.

- Installation will take approximately 5-7 minutes, depending on your systems capabilities
- A dialog will pop-up similar to the one below

CQ5 install/startup dialog

Copyright 2010, Day Software AG, Switzerland

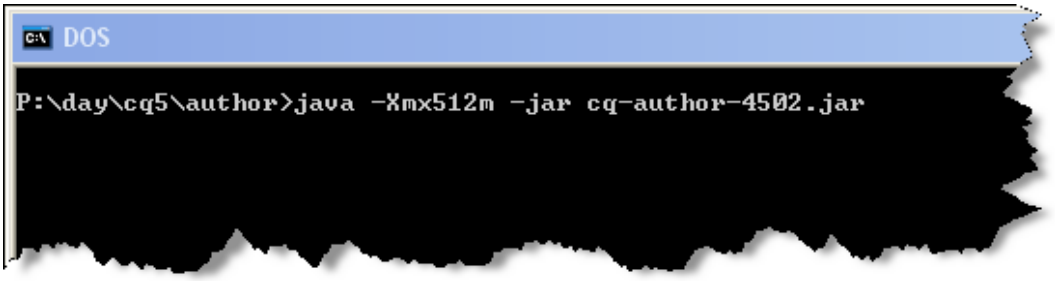
5. Enter the default administrator "Username" (**admin)** and "Password" (**admin**) in the CQ5 login screen after your favorite Web browser pops-up – then select OK.

CQ5 login dialog

Congratulations! You have successfully installed and started CQ5. To start CQ5 in the future, double-click the renamed CQ5 quickstart JAR file (e.g. cq-author-4502.jar).

NOTE

You can also install/start CQ5 from the command line while increasing the Java heap size, which will improve performance. Please see image below.



CQ5 command line start

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Browse Related Application/Server Interfaces

Goal

The following instructions explain how to browse the application/server interfaces associated with a CQ5 installation. This will enable you to use their administrative/configuration capabilities. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance

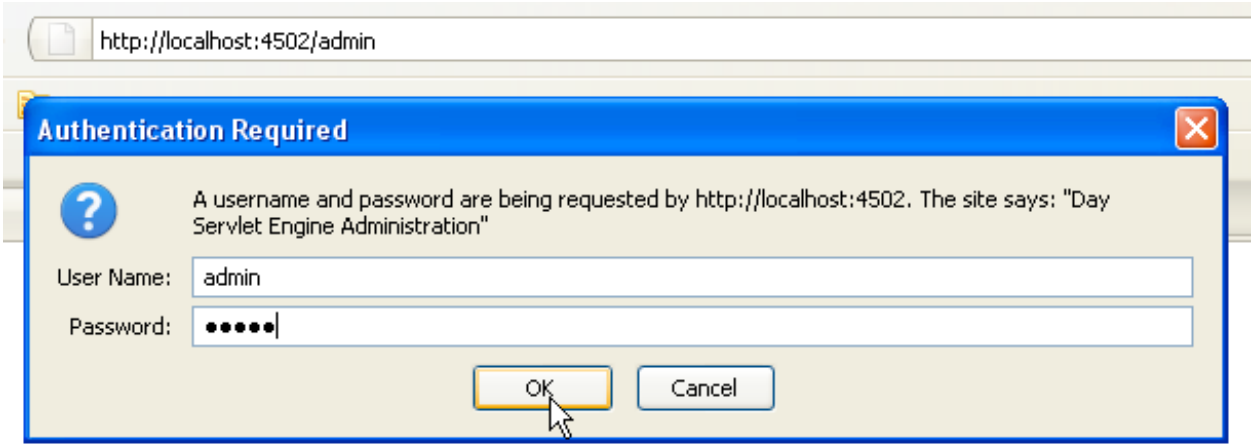
What interfaces exist?

A typical CQ5 installation consists of a Java servlet engine (CQSE), a Java Content Repository (CRX), and a Launchpad (Felix/Sling) application. They each have their own Web interface allowing you to perform expected administrative/configuration tasks.

How to browse the CQSE interface:

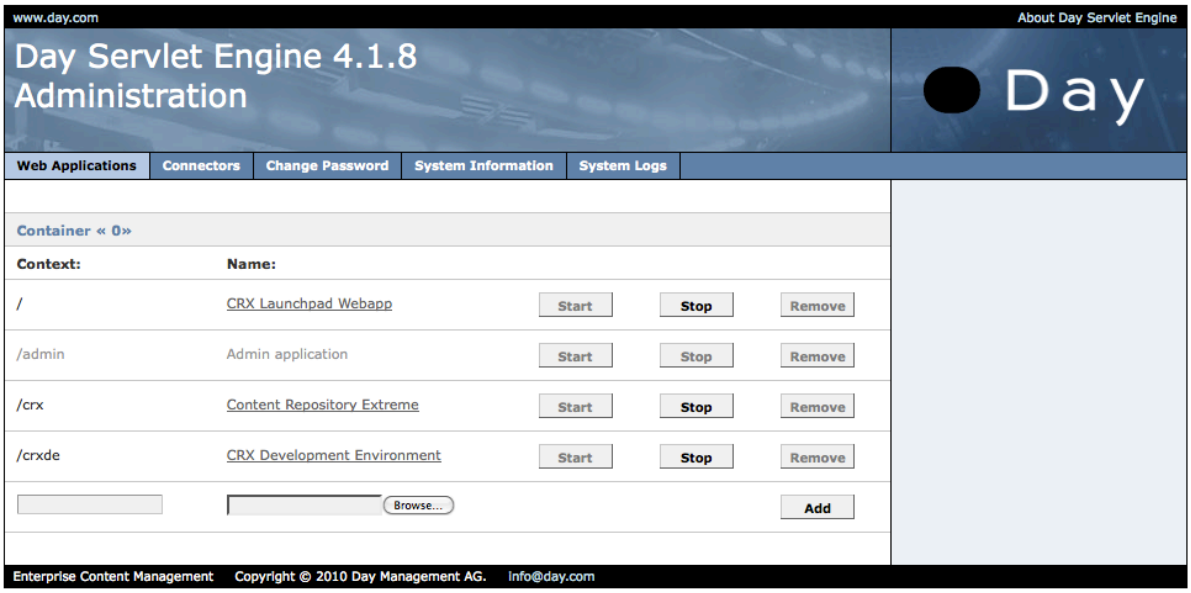
1. Enter the URL http://localhost:4502/admin in your favorite Web browser's address bar.

2. Enter the default administrator "User name" (**admin**) and "Password" (**admin**) in the dialog – then select **OK**.



CQSE login dialog

Copyright 2010, Day Software AG, Switzerland



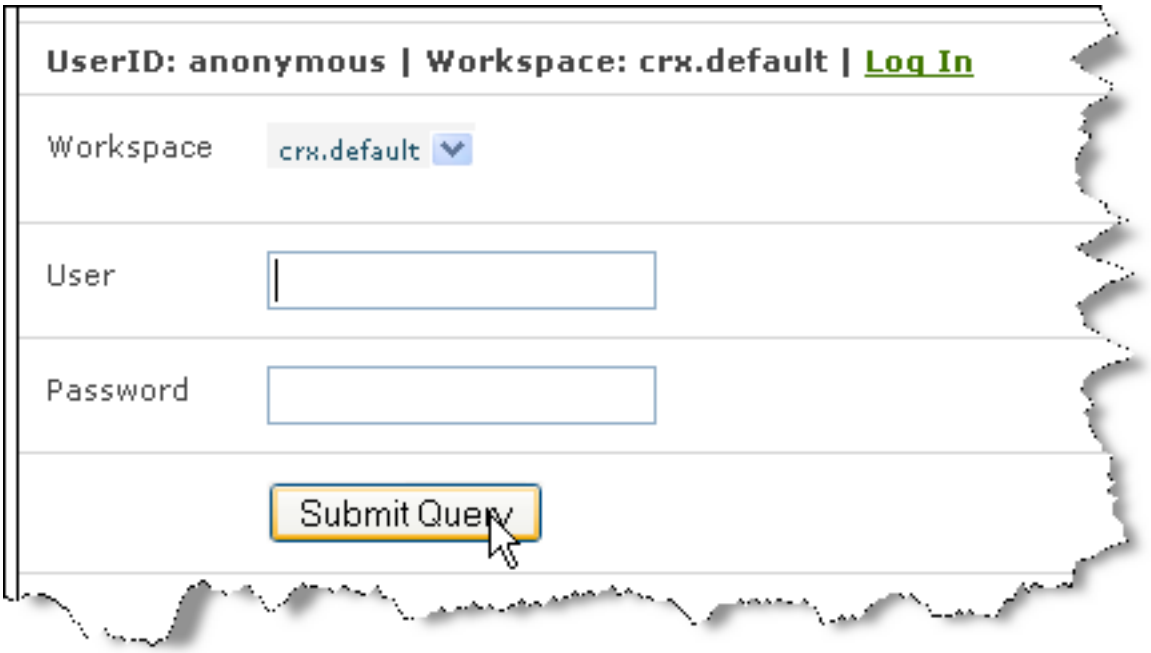
Congratulations! You have successfully logged into the CQSE Java servlet engine. Typically, tasks available through this interface will be handled by the system administrator.

How to browse the CRX interface:

1. Enter the URL http://localhost:4502/crx in your favorite Web browser's address bar – then select Log In after the CRX interface appears.

2. Enter the default administrator "User" (**admin**) and "Password" (**admin**) in the CRX login screen, while continuing to use the crx.default "Workspace" – then select **Submit Query**.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



CRX login screen

3. Select **Content Explorer** in the **Main Console** – a new Web browser window will pop-up.

4. Navigate to the folder */content/geometrixx/en/company* to view a portion of the Geometrixx Web site structure.

CRX content explorer viewing node /content/geometrixx/en/company

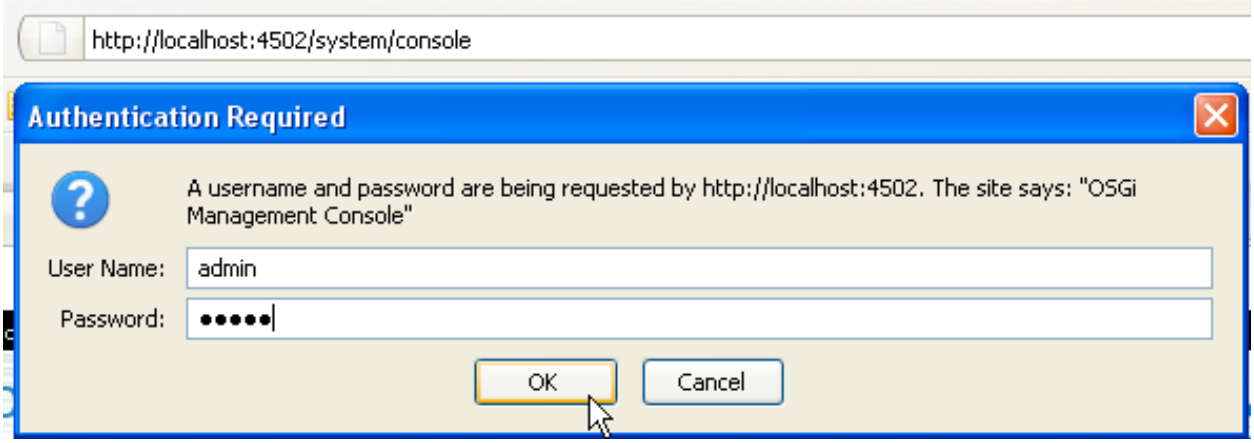
Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully logged into the CRX application and have browsed a portion of the node (Web site) structure. To be a successful developer in CQ5, you need to be able to easily explore/edit nodes and properties at the CRX level.

How to browse the Felix interface:

1. Enter the URL http://localhost:4502/system/console in your favorite Web browser's address bar.

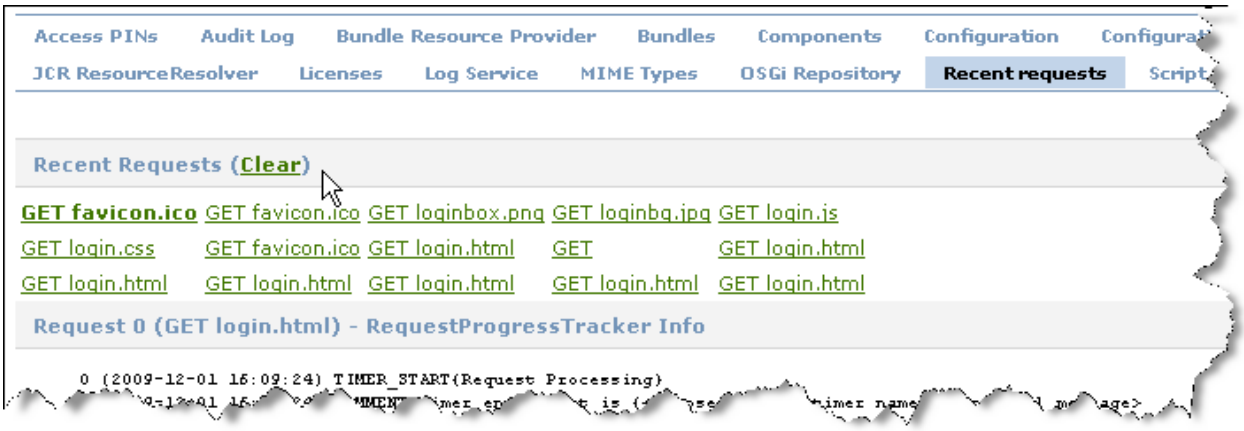
2. Enter the default administrator "User name" (**admin**) and "Password" (**admin**) in the dialog – then select **OK**.



Felix login dialog

3. Select **Recent requests** – then select **Clear** to remove recent requests from the displayed list.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



Felix recent requests

Congratulations! You have successfully logged into the Felix application and have removed recent requests from the displayed list. You can use this request information to investigate the internal workings of CQ5.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create an Application/Project

Goal

The following instructions explain how create an application/project in CQ5 using CRXDE Lite. This will provide an area for you to build CQ5 elements (Templates, Component, etc.) To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance

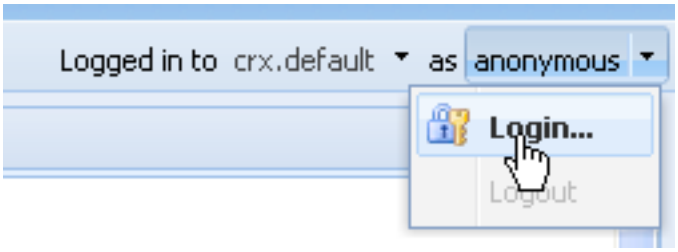
What is CRXDE Lite?

CRXDE Lite is embedded into CQ5/CRX and enables you to perform standard development tasks in a Web browser. With CRXDE Lite, you can create and edit files (e.g. JSP, Java, HTML, etc.), folders, Templates, Components, Dialogs, nodes, properties, and bundles; all while logging and integrating with SVN. CRXDE Lite is recommended when you do not have direct access to the CQ5/CRX server, when you develop an application by extending or modifying the out-of-the-box Components and Java bundles, or when you do not need a dedicated debugger, code completion and syntax highlighting.

How to use CRXDE Lite:

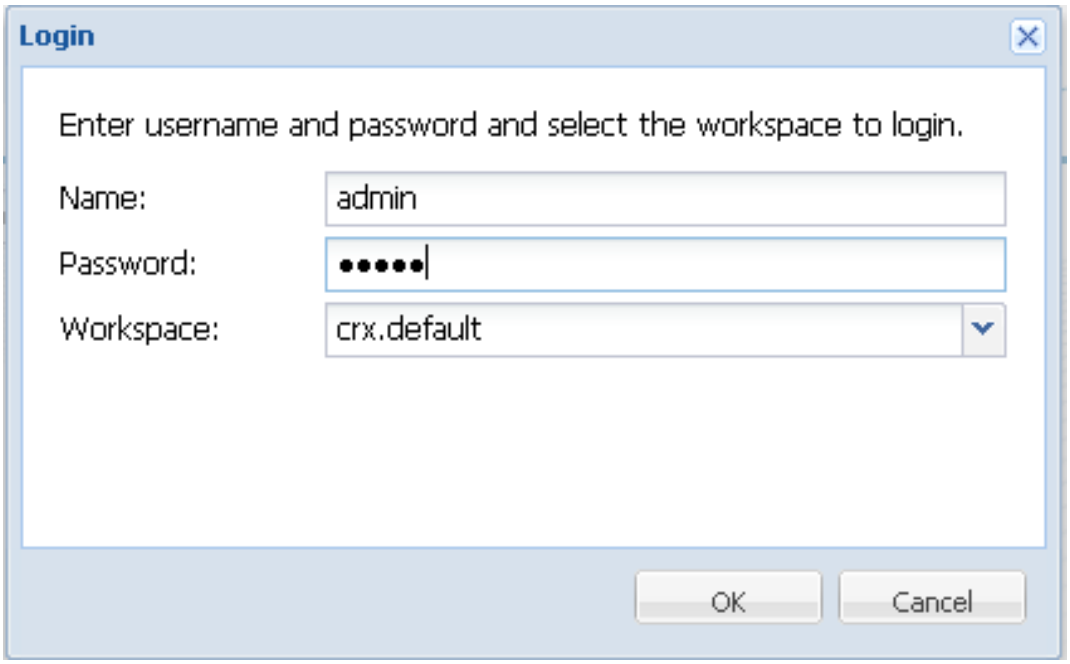
1. Enter the URL http://localhost:4502/crxde in your favorite Web browser's address bar.

2. Select **Login** – then enter the default administrator "Name" (**admin**) and "Password" (**admin**) in the dialog, while continuing to use the crx.default "Workspace" – then select **OK**.



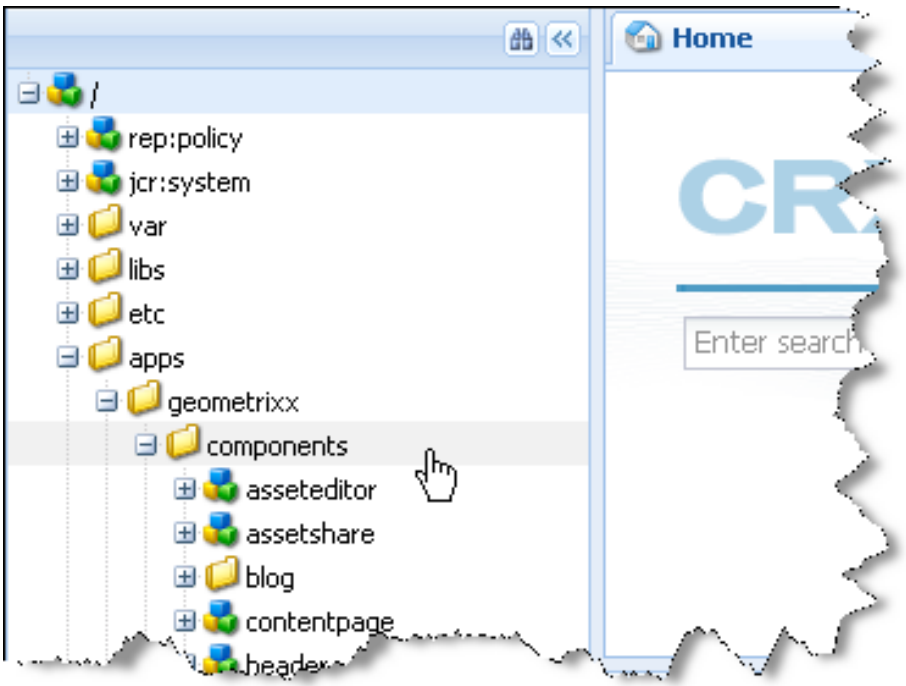
CRXDE Lite login selection

Copyright 2010, Day Software AG, Switzerland



CRXDE Lite login dialog

3. Navigate to the folder */apps/geometrixx/components* to view the custom components created for the Geometrixx Web site/project.



CRXDE Lite geometrixx component display

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully logged into CRXDE Lite and have browsed the custom components created for the Geometrixx Web site/project. Again, CRXDE Lite is embedded into CQ5/CRX and enables you to perform standard development tasks in a Web browser.

What is an application/project?

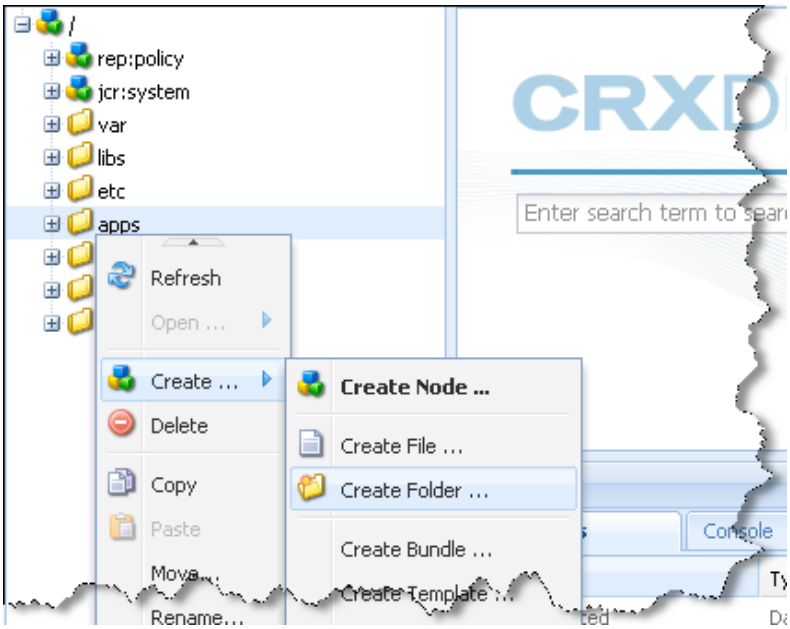
An application/project is where you will store CQ5 elements such as Templates, Components, OSGi bundles, and static files. In order to start a new application/project, it is necessary to define a location for these elements. Typically, they are defined as a subfolder of the /apps folder. Day recommends that you create the following structure for your application/project:

Structure	Description
/apps/<application name>	The application container
/apps/<application name>/components	The Components container
/apps/<application name>/components/page	The “Page” Components container
/apps/<application name>/components/content	The “Content” Components container
/apps/<application name>/templates	The Templates container
/apps/<application name>/src	The OSGi bundles container
/apps/<application name>/install	The compiled OSGi bundles container
/apps/<application name>/global	The static and other global files container

Copyright 2010, Day Software AG, Switzerland

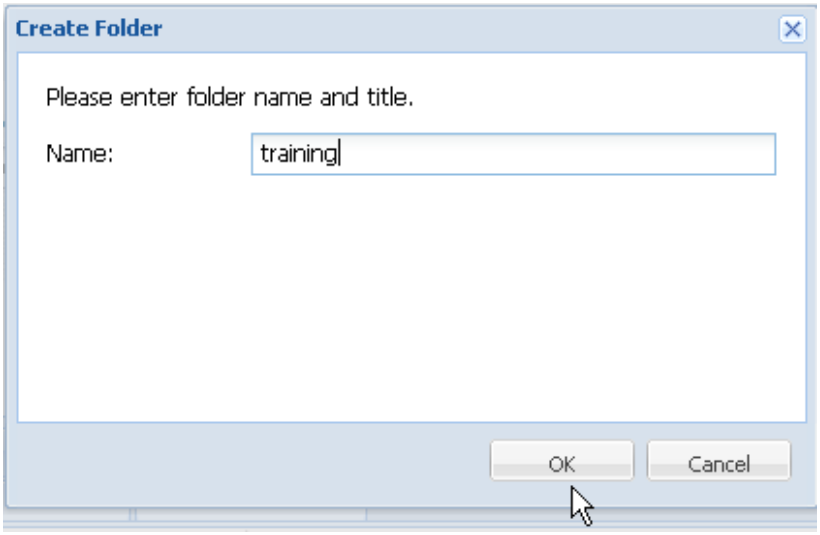
How to create a project using CRXDE Lite:

1. Right-click the folder under which you want to create the new folder – then select **Create ..., Create Folder ...**



CRXDE Lite create folder selection

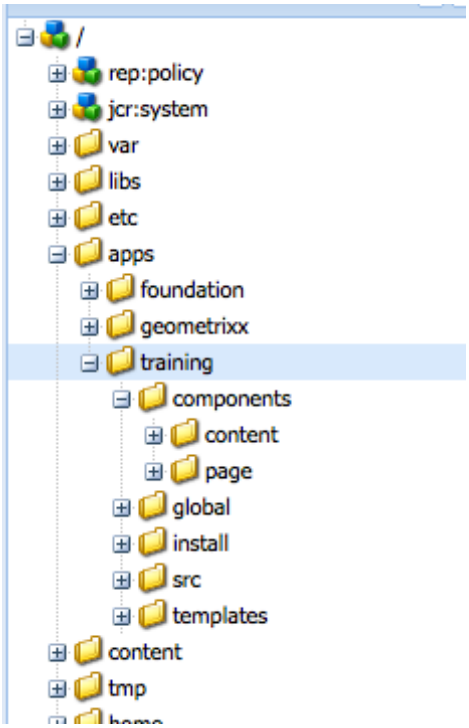
2. Enter the folder "Name" (***training***) you wish to create in the dialog – then select **OK.**



CRXDE Lite create folder dialog

Copyright 2010, Day Software AG, Switzerland

3. Repeat this process until you have created Day's recommended application/project structure – then click **Save All.**



CRXDE Lite completed application structure

CRXDE Lite save all

Congratulations! You have successfully created an application/project and related structure in CQ5. It is not uncommon to add additional structure to the components folder, as there can be many types of custom Components (e.g. nav, header, footer, etc.).

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create a Template

Goal

The following instructions explain how to create a Template using CRXDE Lite. By creating a custom Template, you will be able to create Pages based off of that Template, enforcing a consistent look and feel. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- Basic familiarity with CRXDE Lite
- An application/project structure where you will create the Template

What is a Template?

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 use, and allow access to, Widgets, which are used to author/render content.

A Template is the basis of a Page. To create a Page, the Template's content must be copied (/apps/<application name>/templates/<template name>) to the corresponding position in the site-tree (this occurs automatically if page is created using CQ5).

This copy action also gives the Page its initial content and the property sling:resourceType, the path to the "Page" Component that is used to render the page.

How to create a Template:

1. Right-click */apps/<application name>/templates* – then select **Create..., Create Template ...**

2. Enter the desired Template “Label”, “Title”, “Description”, “Resource Type”, and “Ranking” in the dialog – then select **Next.**

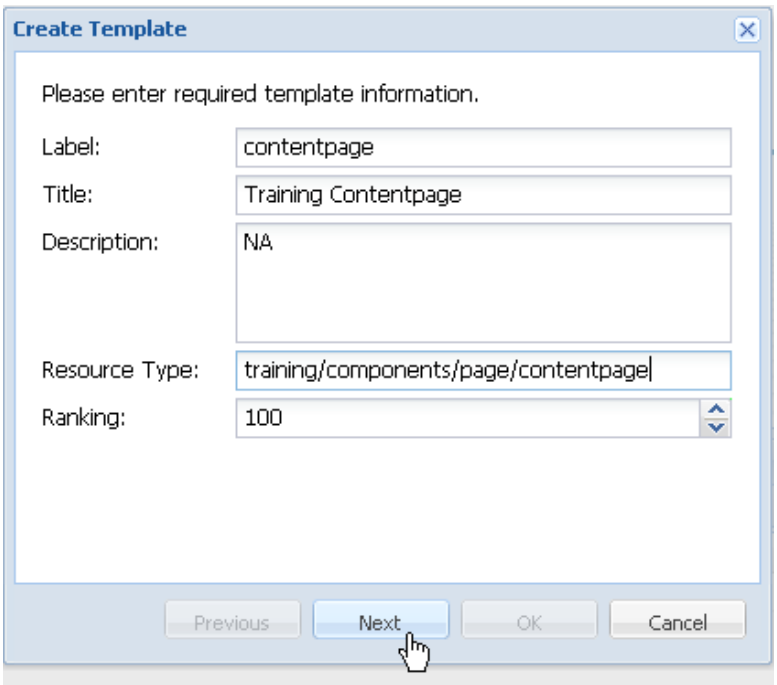
Copyright 2010, Day Software AG, Switzerland

- Label = the name of the Template/node that will be created
  - contentpage
- Title (property jcr:title) = the title that will be assigned to the Template
  - Training Contentpage
- Description (property jcr:description) = the description that will be assigned to the Template
  - NA
- Resource Type (property sling:resourceType) = the Component's path that will be assigned to the Template and copied to implementing pages
  - training/components/page/contentpage

  NOTE

  The property ***sling:resourceType*** will be created on the Template's ***jcr:content*** node. In addition, the Component may not yet exist. This is because you have not yet created it.

- Ranking (property ranking) = the order (ascending) in which this Template will appear in relation to other Templates. Setting the Rank to ***1***, will ensure that our Template appears first in the list.
  - 1



CRXDE Lite create template dialog

Copyright 2010, Day Software AG, Switzerland

3. Select **Next** for "**Allowed Paths**" . Allowed Paths will define paths where this template may be used to create pages.

CRXDE Lite create template allowed paths

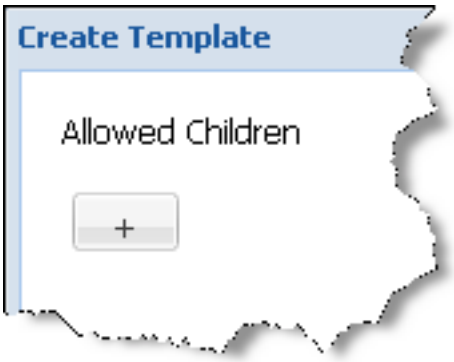
4. You will need to add a path to **Allowed Paths**. Click on the plus sign and enter the following value: */content(/.*)?*

NOTE

If you forget to enter allowedPaths before clicking on Finish. Create the following property on your template node:

- Name: allowedPaths
- Type: String[] (String Array)
- Value: /content(/.*)?

5. Click **Next** for "**Allowed Parents**" – then select **OK** on "**Allowed Children**".



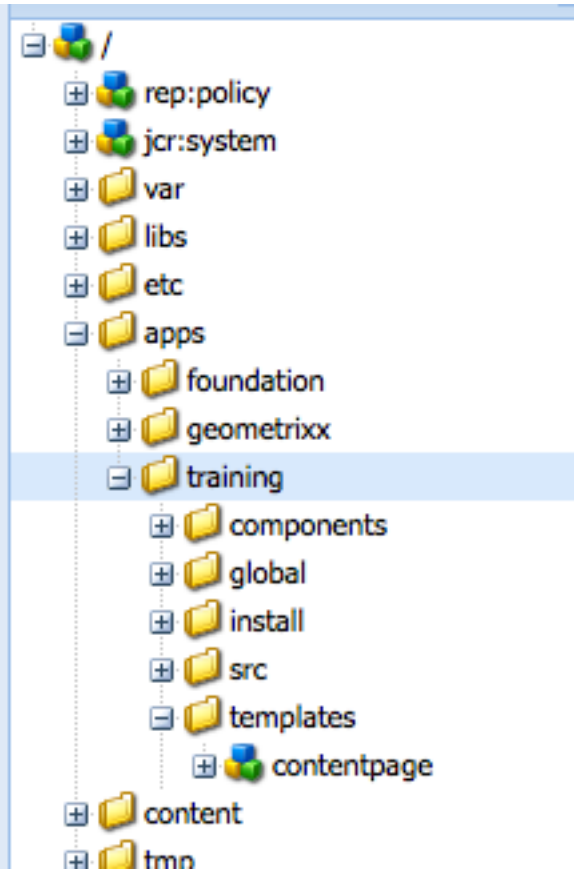
CRXDE Lite create template allowed children

Copyright 2010, Day Software AG, Switzerland

NOTE

Don't forget to click **Save All** to persist your changes in the repository.

Congratulations! You have successfully created a Template in CQ5. To make you more aware of the structure associated with CQ5 objects, it is a good idea to view the nodes and properties that were created in the repository (CRX). Please see the image below.



CRXDE lite view of template

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create a "Page" Component

Goal

The following instructions explain how to create a “Page” Component using CRXDE Lite. The Template you created in a previous exercise will use this Component's resources (scripts, dialogs, etc.) to aid in rendering Pages based off of that Template. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- Basic familiarity with CRXDE Lite
- An application/project structure where you will create the Component

What is a "Page" Component?

Components are modular, re-usable units that implement specific functionality/logic to render the content of your Web site. They have no hidden configuration files, can include other Components, and can run anywhere within CQ5 or in isolation (e.g. portal). A Component could be described as a collection of scripts (e.g. JSPs, Java servlets, etc.) that completely realize a specific function. More specific, a "Page" Component is typically referenced by a Template.

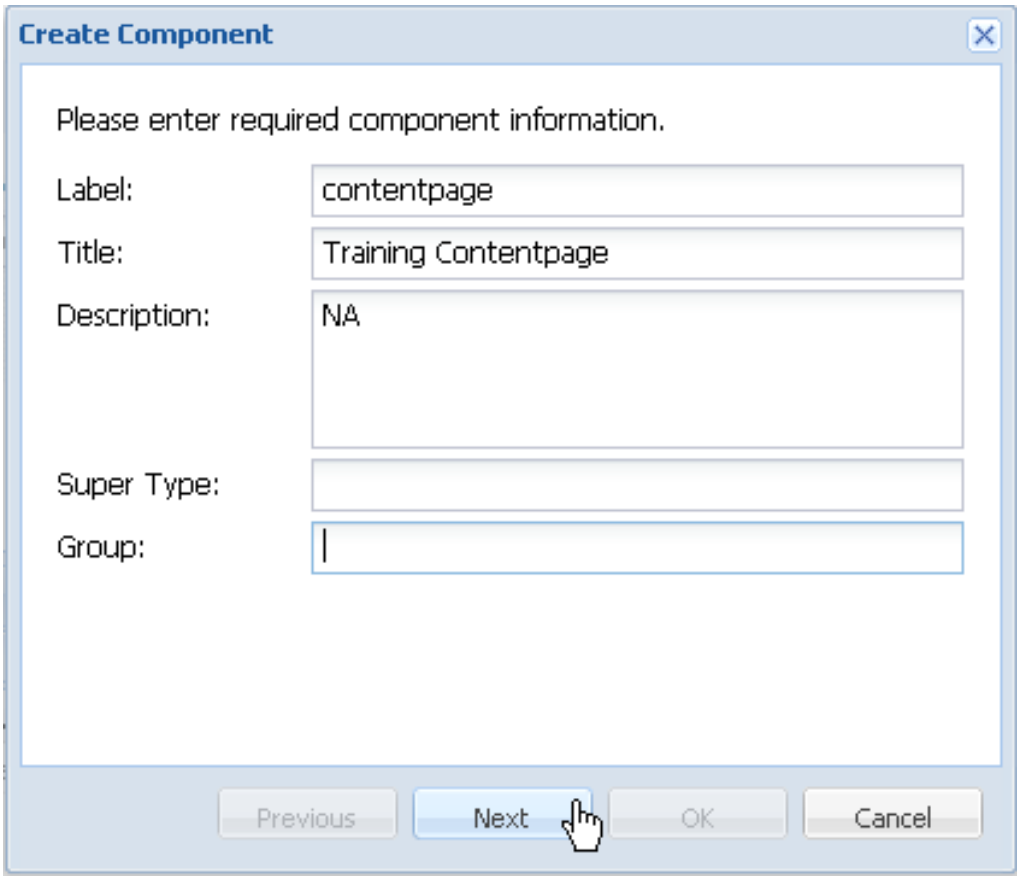
How to create a “Page” Component using CRXDE Lite:

1. Right-click */apps/<application name>/components/page* – then select **Create..., Create Component ...**

2. Enter the desired Component “Label”, “Title”, and “Description” in the dialog – then select **Next**.

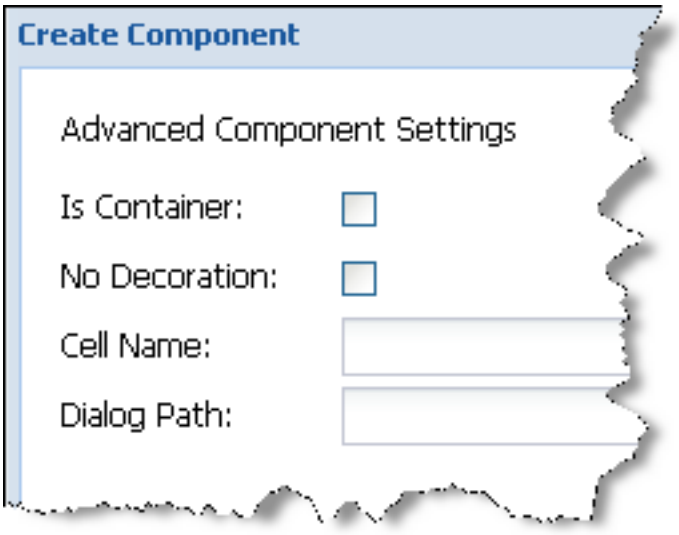
- Label = the name of the Component/node that will be created
  - contentpage
- Title (property jcr:title) = the title that will be assigned to the Component
  - Training Contentpage
- Description (property jcr:description) = the description that will be assigned to the Component
  - NA

Copyright 2010, Day Software AG, Switzerland



CRXDE Lite create component dialog

3. Select **Next** for "**Advanced Component Settings**" and "**Allowed Parents**" – then select **OK** on "Allowed Children".



CRXDE Lite create component advanced settings

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

A Component could be described as a collection of scripts (e.g. JSPs, Java servlets, etc.) that completely realize a specific function. In order to realize this specific functionality, it is your responsibility to create any necessary scripts that will do so. Typically, a Component ("Page" or otherwise) will have at least one default script, identical to the name of the Component (e.g. contentpage.jsp). Notice that the creation of the component results in the creation of the default **contentpage.jsp** script.

4. Open the script by double-clicking the script object in the left pane. The script contains some sample code that may have to be adjusted or deleted, depending on what the component will do.

5. Enter some HTML code, similar to below – then select **Save All**.

contentpage.jsp

```
<html>
    <head>
        <title>Hello World !!!</title>
    </head>
    <body>
        <h1>Hello World !!!</h1>
        <h2>Welcome to a new Day</h2>
    </body>
</html>
```

NOTE

Don't forget to click **Save All** to persist your changes in the repository.

Congratulations! You have successfully created a "Page" Component in CQ5. You can now create a script that will render content based on your requirements. In addition, you will perform a similar process when creating "Content" Components in the future.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create Pages & Web Site Structure

Goal

The following instructions explain how to create Pages and Web site structure in CQ5. These Pages are what/where author's will manage content. To successfully complete and understand these instructions, you will need:

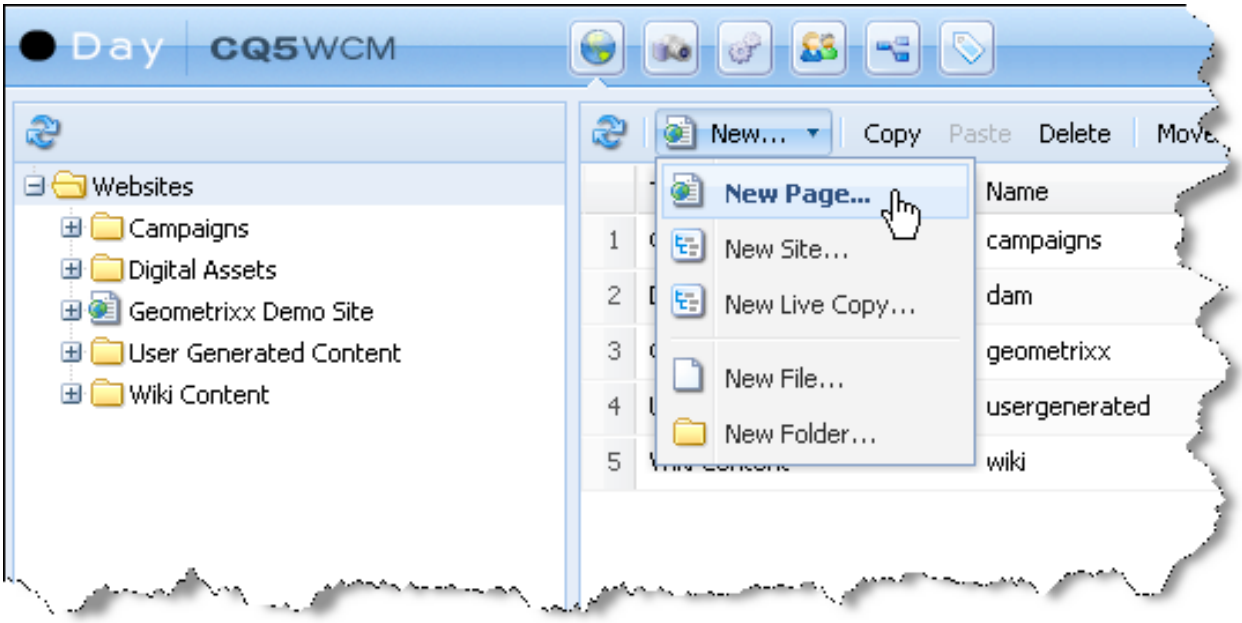
- A running CQ5 Author instance
- A completed "Page" Component

What is a Page?

A Page is where content authors will create and edit content that will most likely be published and viewed by site visitors. It is an exact copy of the Template from which it was created.

How to create a Page:

1. Select "**Websites**" in the CQ5 Siteadmin tool – then select **New ..., New Page ...**

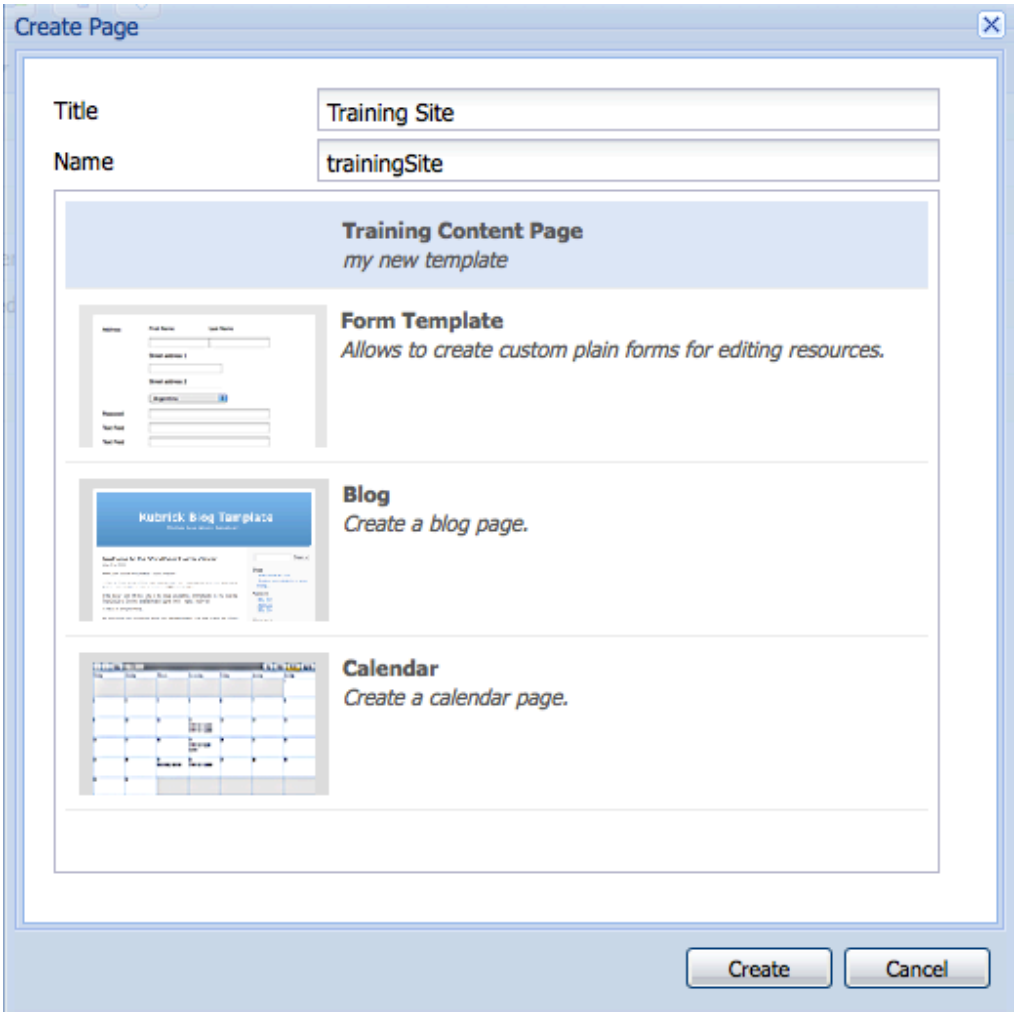


CQ5 siteadmin new page selection

Copyright 2010, Day Software AG, Switzerland

2. Enter and select the desired Page “Title”, “Name” and Template on which to base this Page on – then click **Create**.

- Title (property jcr:title) = the title that will be assigned to the Page
- Name = the name of the Page/node that will be created



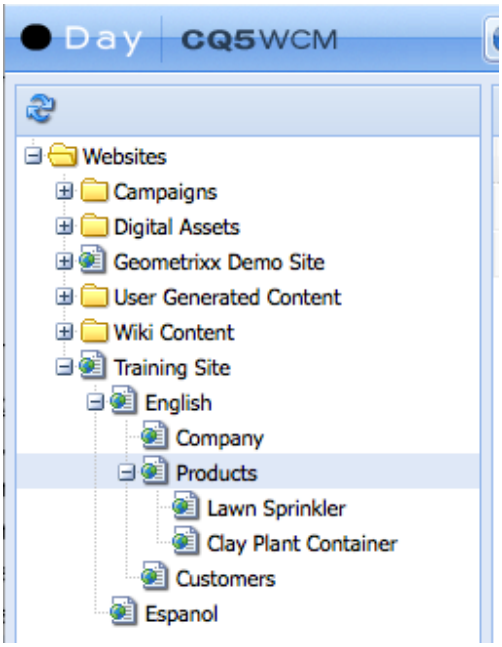
CQ5 siteadmin new page dialog

3. Open the newly created Training Site page by double-clicking it in the right pane – then view its output to ensure it meets your expectations.

Copyright 2010, Day Software AG, Switzerland

CQ5 page view (e.g. /content/trainingSite)

4. Repeat this process until you have created a Web site structure similar to the image below.



CQ5 siteadmin web site structure

NOTE

Since all Pages will be created using the same Template (i.e. Training Contentpage) and implement the same rendering script, every Page will look identical – for now.

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully created Pages and Web site structure in CQ5. This is the type of behavior you can expect from content authors when building out their Web site structure.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Install & Start CRXDE

Goal

The following instructions explain how to install and start CRXDE, Day's Eclipse based IDE application. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A CRXDE package (ZIP file)

What is CRXDE?

CRXDE is a pre-packaged stand-alone Eclipse application. CRXDE is custom-built specifically for CQ and CRX and thus enables you to efficiently develop your project. CRXDE gives you a broad set of tools to easily create and manage files, folders, Templates, Components, Dialogs, nodes, properties, scripts and OSGi bundles while logging, debugging and integrating with SVN. CRXDE is built on the Eclipse Rich Client Platform (RCP), leveraging the Eclipse File System (EFS) API.

How to install and start CRXDE:

1. Create/identify a folder structure on your file system where you will store, install, and start CRXDE (e.g. C:/day/cq5).

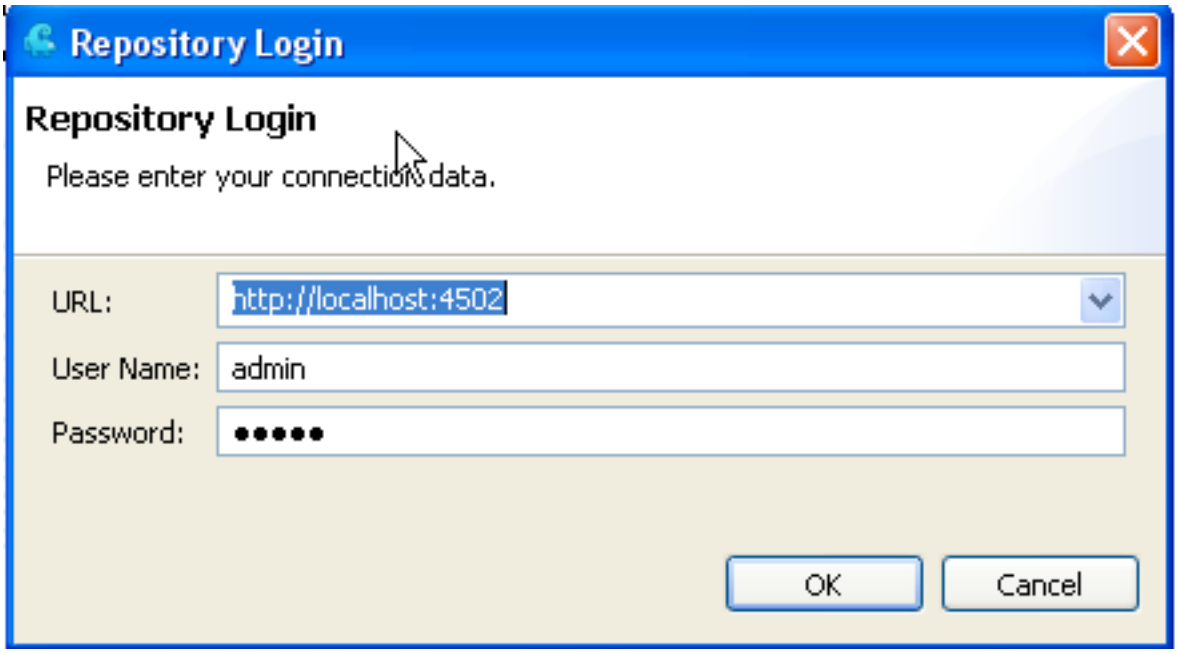
2. Copy the appropriate CRXDE package from <USB>/distribution/crxde into the folder structure.

3. Extract the package.

4. Double-click the executable in the CRXDE folder.

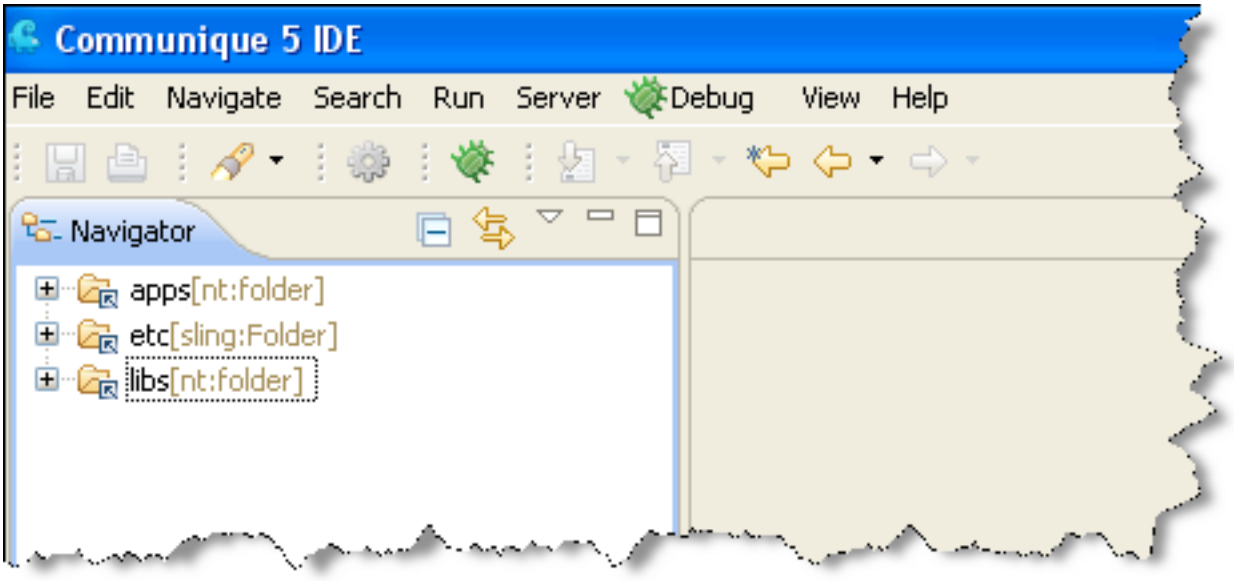
5. Enter the "URL" (http://localhost:4502), default administrator "User Name" (**admin**) and "Password" (**admin**) in the dialog – then click OK.

Copyright 2010, Day Software AG, Switzerland



CRXDE login dialog

Congratulations! You have successfully installed and started CRXDE. Here you will be able to create and modify CQ5 elements such as Templates, Components, scripts, etc. In addition, traditional IDE tools such as code completion and debugging will now be available to you. Below is a view of what CRXDE looks like after starting.



CRXDE interface view

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Utilize CRXDE

Goal

The following instructions explain how to use CRXDE to perform basic development tasks. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage)

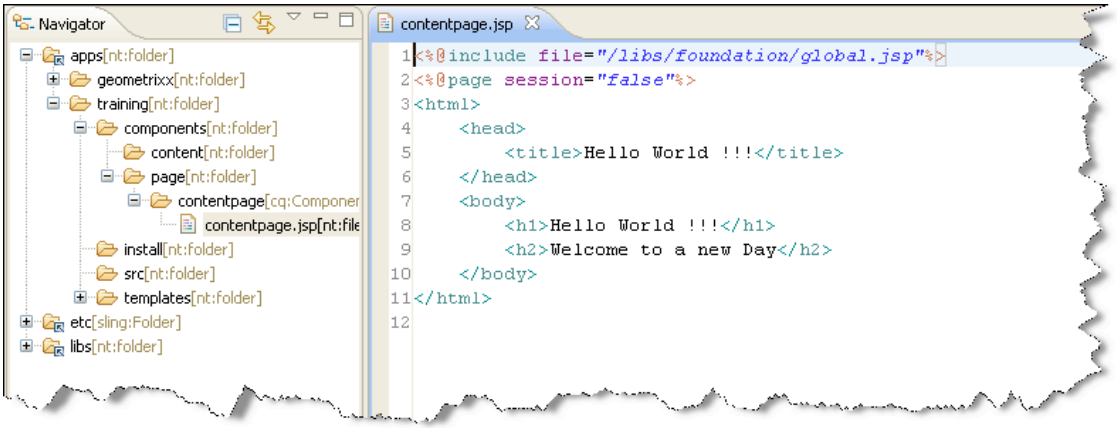
What are the benefits of using CRXDE vs. CRXDE Lite?

As stated earlier, CRXDE Lite is recommended when you do not have direct access to the CQ5/CRX server, when you develop an application by extending or modifying the out-of-the-box Components and Java bundles, or when you do not need a dedicated debugger, code completion and syntax highlighting. CRXDE is recommended when you are developing complex applications by creating new components and Java bundles. Again, CRXDE hides some of the complexity of the development process by allowing you to work directly with the repository without the need to synchronize the repository with the file system.

How to use CRXDE:

1. Navigate to and open, by double-clicking, the contentpage.jsp script you created earlier for the Training Contentpage “Page” Component (e.g. /apps/training/components/page/contentpage).

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



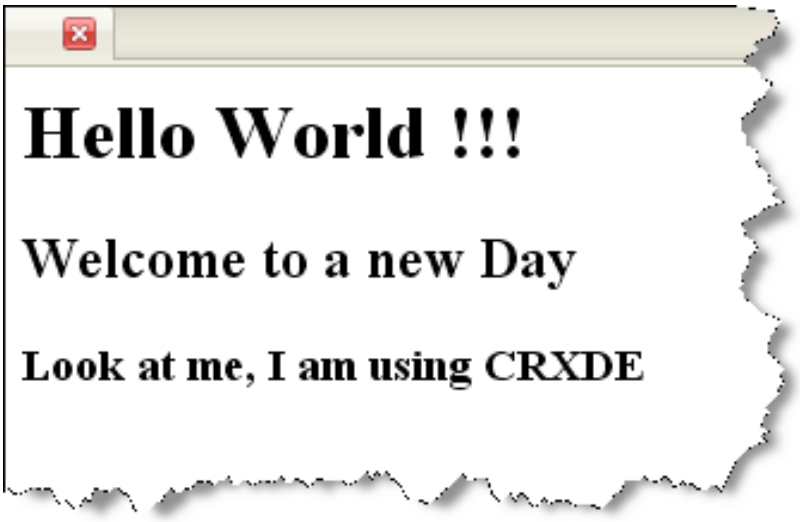
CRXDE view of contentpage.jsp

2. Edit the script to your liking – then click **File, Save.**

NOTE

Traditional keyboard shortcuts will work in CRXDE (e.g. Ctrl-S to save, Ctrl-C to copy, etc.).

3. In the CQ5 Siteadmin tool, navigate to and open any page that is based off of this "Page" Component to view your changes.



CQ5 page view

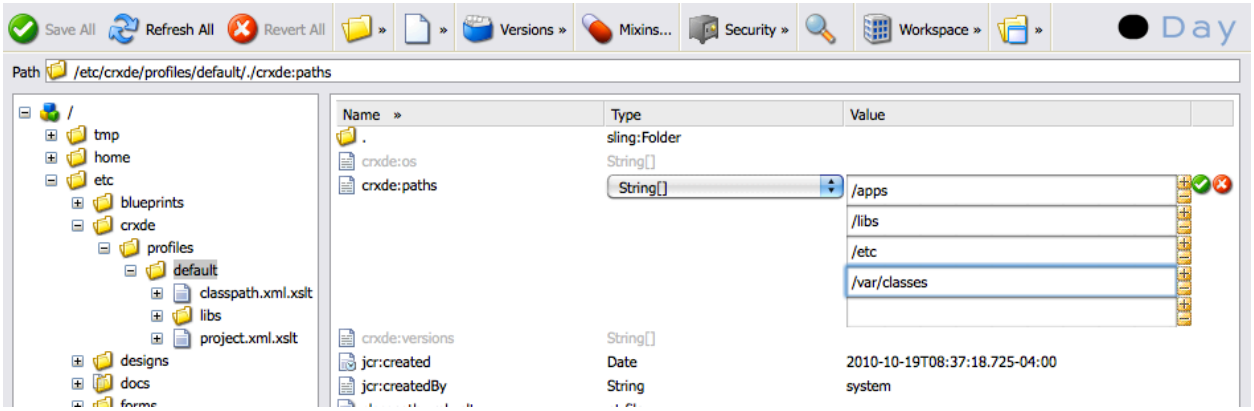
Congratulations! You have successfully modified a script using CRXDE. For the rest of this training, you will use CRXDE to accomplish more complex development tasks.

Copyright 2010, Day Software AG, Switzerland

You will notice that the tree browser in CRXDE includes only /apps, /etc, and /libs. If you want to see additional branches of the repository, similar to CRXDE Lite, you can use the CRX Content Explorer or CRXDE Lite to export the additional nodes to CRXDE.

How to modify the set of repository paths exported to CRXDE using the CRX Content Explorer:

1. Navigate to http://localhost:4502/crx.
2. Login if you have not already logged in.
3. Select the Content Explorer.
4. In the Content Explorer, navigate to */etc/crxde/profiles/default*
5. Modify the *crxde:paths* property by adding desired paths. (e.g., */var/classes*)



6. Click the green check mark.
7. Save All.

Congratulations! You have successfully modified the paths exported to CRXDE.

Copyright 2010, Day Software AG, Switzerland

You can also add Java API documentation, for CQ5 APIs and your own APIs, to CRXDE. Simply define the location of the javadocs to CRXDE using the Package Explorer pane.

How to add java API documentation to the CRXDE:

1. In CRXDE, switch the view from Navigator to Package Explorer.
2. Expand the localhost_4502 folder and then expand the Reference Libraries folder.
3. Right-click *cq-wcm-api-5.3.2.jar* and select Properties from the context menu.
4. Select Javadoc Location.
5. Use the browse button to navigate to and select the location of the unpacked API documentation. (e.g, <cq-install-dir>/crx-quickstart/docs

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Include the "global.jsp" in the Page Component

Goal

The following instructions explain how to include an existing CQ5 "power" script. This will give you access to numerous development objects, at compilation time. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage)

Why include the global.jsp script?

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

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

The Day provided *global.jsp* 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.

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).

Copyright 2010, Day Software AG, Switzerland

- 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).
- currentNode
  - the current JCR node object (javax.jcr.Node 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).
- resource
  - the current Sling resource object (org.apache.sling.api.resource.Resource interface).
- 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).

Copyright 2010, Day Software AG, Switzerland

How to include the global.jsp script in your Training "Page" Component:

1. Open the **contentpage.jsp** script by double-clicking it.

2. Enter the include statement in your JSP, similar to below – then Save.

contentpage.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <head>
        <title>Hello World !!!</title>
    </head>
    <body>
        <h1>Hello World !!!</h1>
        <h2>Welcome to a new Day</h2>
        <h3>Look at me, I am using CRXDE</h3>
    </body>
</html>
```

3. Test your script by requesting a Page in CQ5 Siteadmin that implements this "Page" Component.

- If successful, you should not notice any difference in the way the Page is rendered

Congratulations! You have successfully included the global.jsp, allowing you access to numerous Sling, CQ, and Java objects to aid you in your development efforts.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Display Basic Page Content

Goal

The following instructions explain how to dynamically display basic Page content such as Page title, name, and path. This is an introduction of how to render content that lives in the repository, which is a similar concept to querying and displaying data from a database. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

How can I display basic Page content?

After inclusion of the global.jsp in whatever script you are working on, there are generally three ways to access content in CQ5:

- Via the properties object
  - The properties object is an instance of the ValueMap (see Sling API) class and contains all properties of the current resource. For example:

```
...
String pageTitle = properties.get("jcr:title", "NO
TITLE");
...
```

- Via the currentPage object
  - The currentPage object is an instance of the Page (see CQ5 API) class, which provides some methods to access content. For example:

```
...
String pageTitle = currentPage.getTitle();
...
```


Copyright 2010, Day Software AG, Switzerland

- Via currentNode object
  - The currentNode object is an instance of the Node (see JCR API) class, which provides access to content via the getProperty() method. For example:

```
...
String pageTitle = currentNode.getProperty
("jcr:title").getString();
...
```

NOTE

Day does not recommend using the JCR API directly in CQ5 unless necessary. Since CQ5 is a Sling application, you should deal with resources and not JCR nodes.

How to dynamically display basic Page content in your Training "Page" Component:

1. Open the contentpage.jsp script.

2. Enter some JSP and HTML code, similar to below – then **Save**.

contentpage.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <head>
        <title><%= currentPage.getTitle() == null ? currentPage.getName() :
currentPage.getTitle() %></title>
    </head>
    <body>
        <h2>properties</h2>
        Title: <%= properties.get("jcr:title") %><br />

        <h2>currentPage</h2>
        Title: <%= currentPage.getTitle() %><br />
        Name: <%= currentPage.getName() %><br />
        Path: <%= currentPage.getPath() %><br />
        Depth: <%= currentPage.getDepth() %><br />

        <h2>currentNode</h2>
        Title: <%= currentNode.getProperty("jcr:title").getString() %><br />
        Name: <%= currentNode.getName() %><br />
        Path: <%= currentNode.getPath() %><br />
        Depth: <%= currentNode.getDepth() %><br />
    </body>
</html>
```


Copyright 2010, Day Software AG, Switzerland

3. Test your script by requesting a Page in CQ5 Siteadmin that implements this "Page" Component.

- If successful, you should now see content related to the requested Page being displayed dynamically, similar to the image below.

Page basic content output

Congratulations! You have successfully displayed content related to a Page dynamically. This is a monumental step as this is a large portion of what you do as a developer – display content.

Copyright 2010, Day Software AG, Switzerland

Day 2
EXERCISE - Create Multiple Scripts/Renderers for the "Page" Component

Goal

The following instructions explain how to create and use multiple scripts/renders for a single Component. This will allow for multiple views of the same content, which is a huge proponent for using Sling. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage)

What is Sling and why have multiple scripts/renderers for a Component?

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.

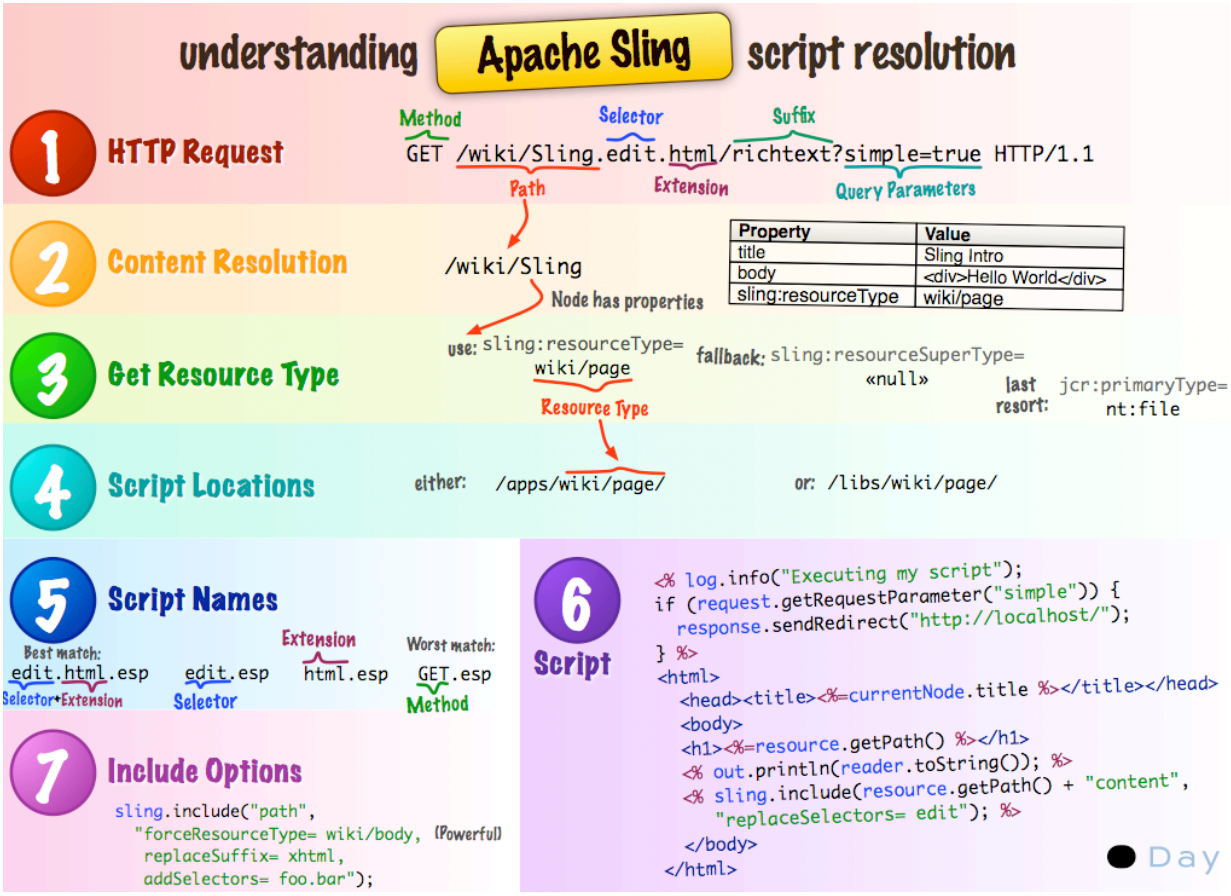
Sling started as an internal project of Day Management AG and is included in the installation of CQ5. Sling has since been contributed to the Apache Software Foundation - further information can be found at Apache (http://sling.apache.com).

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/viewed differently. The following diagram explains the Sling script resolution. It shows how to get from HTTP request to content node, from

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

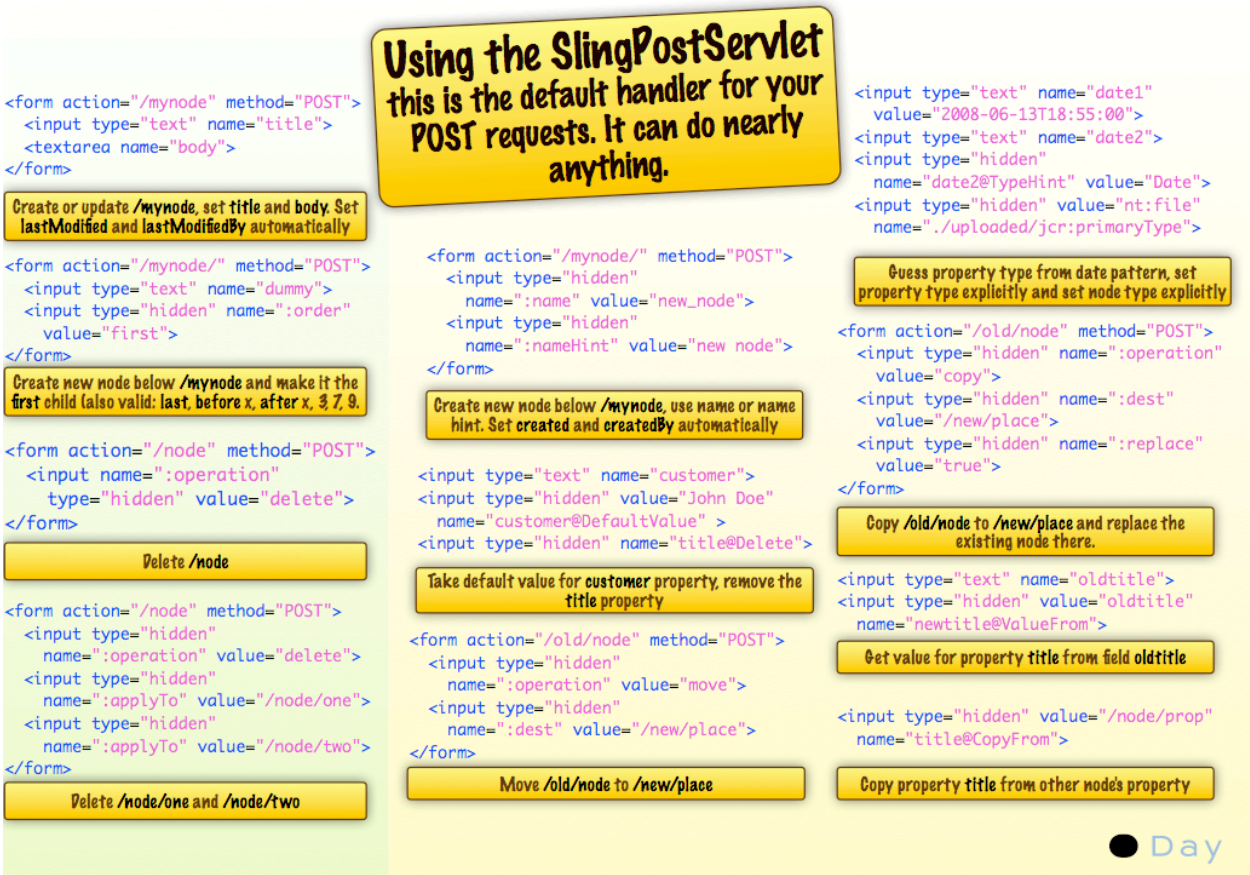
content node to resource type, from resource type to script and what scripting variables are available.



Sling cheat sheet - side 1

The following diagram explains all the hidden, but powerful request parameters you can use when dealing with the SlingPostServlet, the default handler for all POST requests that gives you endless options for creating, modifying, deleting, copying and moving nodes in the repository.

Copyright 2010, Day Software AG, Switzerland



Sling cheat sheet - side 2

In Sling, and therefore also CQ5, processing is driven by the URL of the HTTP 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.print.a4.html/a/b?x=12

We can break it down into its composite parts:

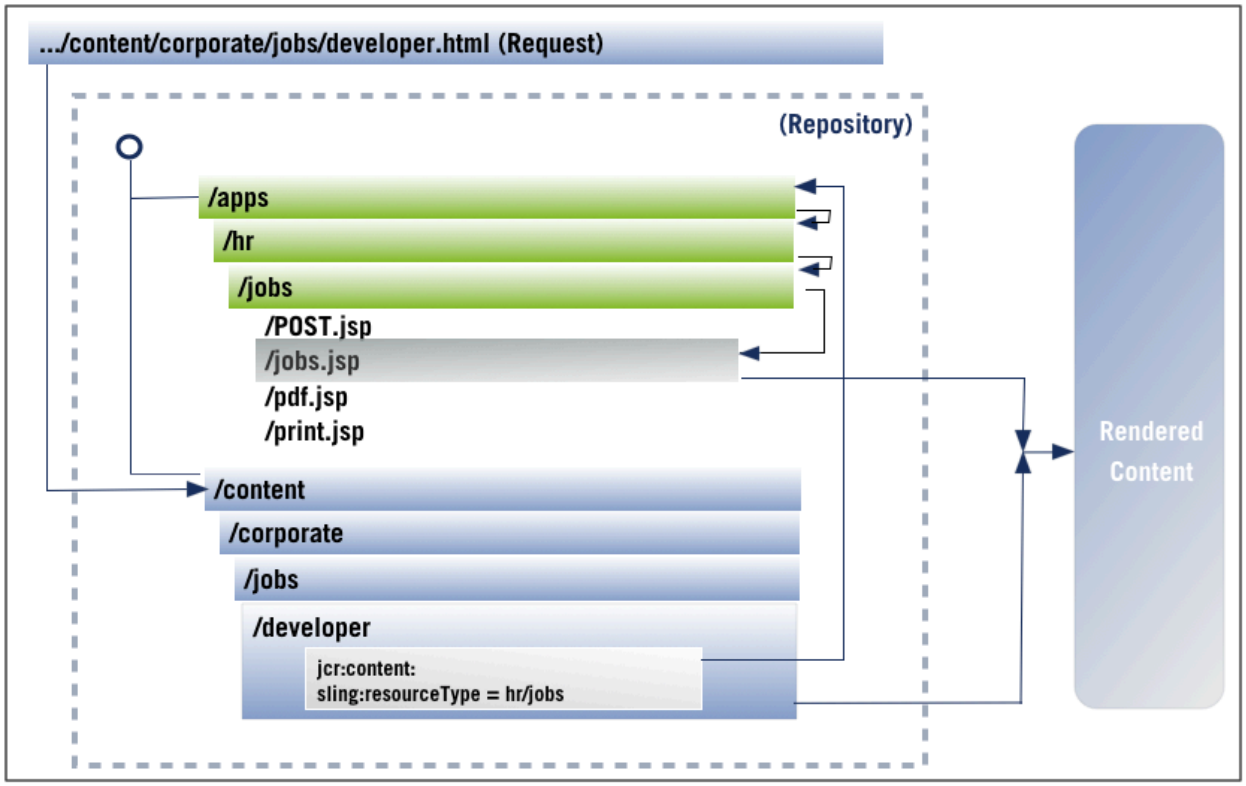
protocol	host	content path	selector	extension		suffix		param(s)
http://	myhost/	tools/spy	.print.a4	.html	/	a/b	?	x=12

Sling request decomposition table

Copyright 2010, Day Software AG, Switzerland

When the appropriate resource (content node) is located, the 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 property can be either absolute or relative to a configuration parameter. 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. The following diagram illustrates how a script resolution is resolved.

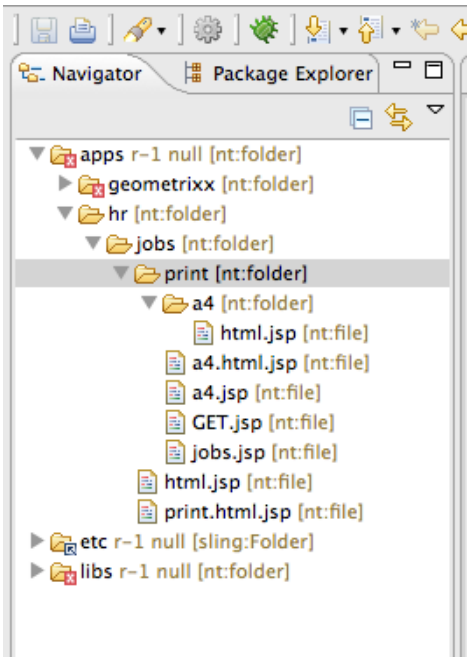


Sling request decomposition

With Sling, you specify which script renders a certain entity (by setting the sling:resourceType property in the jcr:content node). This mechanism offers more freedom than one in which the script accesses the data entities (as an SQL statement in a PHP script would do) as a resource can have several renditions.

Copyright 2010, Day Software AG, Switzerland

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 locations shown, then the order of preference would be as shown:

1. /apps/hr/jobs/print/a4.html.jsp
2. /apps/hr/jobs/print/a4/html.jsp
3. /apps/hr/jobs/print/a4.jsp
4. /apps/hr/jobs/print.html.jsp
5. /apps/hr/jobs/print.jsp
6. /apps/hr/jobs/html.jsp
7. /apps/hr/jobs/jobs.jsp
8. /apps/hr/jobs/GET.jsp

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

From this it can be seen that:

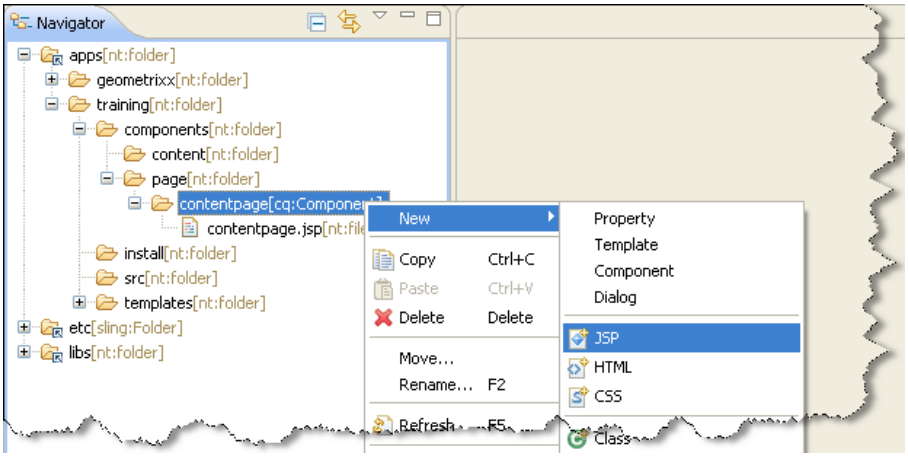
- Folders (i.e. nodes of type nt:folder) take precedence over jsp file names when resolving using selectors.
- Only one selector in a file name has any effect – any files with names containing two selectors don't ever get selected, but names of folders can be used to match the selectors in the request.
- Scripts with HTTP method names (e.g.,GET.jsp) is selected as a last resort, (after the default script: *jobs.jsp*, in this case).
- Scripts with names that include HTTP methods in addition to another selector or extension in a .jsp file name are NEVER selected.

How to create multiple scripts/renderers:

1. Right-click /apps/<application name>/components/page/contentpage -- then select **New, JSP**.

How to create multiple scripts/renderers:

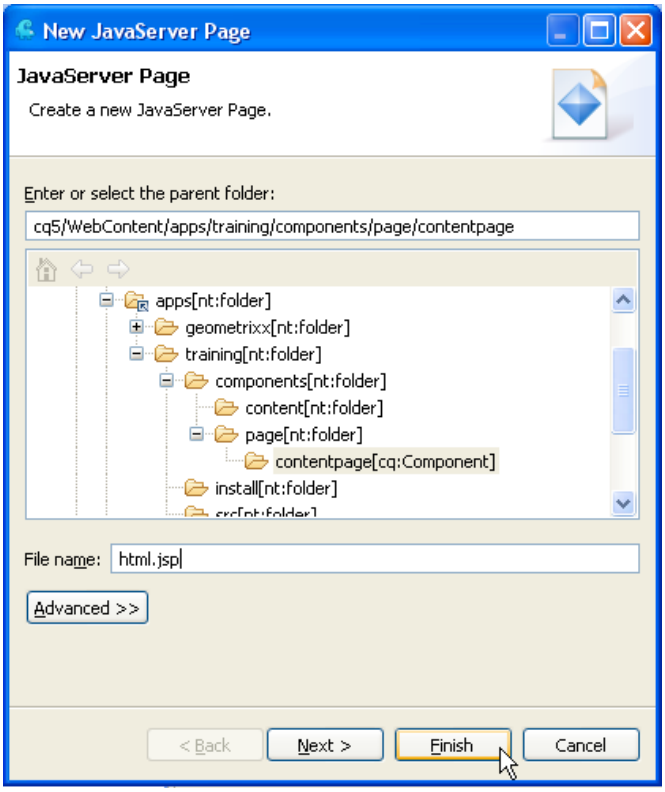
1. Right-click /apps/<application name>/components/page/contentpage -- then select **New, JSP**.



CRXDE new jsp selection

2. Enter the desired file "File name" (**html.jsp**) in the dialog – then select **Finish**.

Copyright 2010, Day Software AG, Switzerland



CRXDE new jsp dialog

NOTE

The newly created script will pop-up in the right pane.

3. Enter some HTML code, similar to below – then **Save**.

html.jsp

```
<html>
    <head>
        <title>Hello World !!!</title>
    </head>
    <body>
        <h1>This is the HTML script/renderer </h1>
    </body>
</html>
```

4. Repeat these processes for a script/renderer named *m.html.jsp*. Change the text in the *m.html.jsp* script to read

Copyright 2010, Day Software AG, Switzerland

```
<h1>This is the MOBILE HTML script/renderer </h1>
```

Remember to **Save**.

5. Test your multiple scripts/renderers by requesting a Page in CQ5 Siteadmin that implements this "Page" Component.

- /content/trainingSite/en/company.html

Page html.jsp script/renderer

- /content/trainingSite/en/company.m.html

Page m.html.jsp script/renderer

Congratulations! You have successfully created multiple scripts/renderers, potentially allowing you to display the same content in two different views.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Breakout/Modularize the "Page" Component

Goal

The following instructions explain how to modularize a Component into multiple scripts and include them at runtime, promoting Component/script reuse. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes

What’s the difference between a JSP, CQ, and Sling include?

The <%@ include file="myScript.jsp" %> directive informs the JSP compiler to include a complete file into the current file – at compilation time. It is as if the contents of the included file were pasted directly into the original file. The <cq:include script="myScript.jsp" /> directive is different in that it includes the file at runtime.

Often the question is asked: "Should I use <cq:include> or <sling:include>?" When developing CQ5 components, Day recommends that you use <cq:include /> tag. This 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.

How to include a script at runtime using <cq:include /> in your “contentpage” Component:

1. Remove (or rename) the scripts *html.jsp* and *m.html.jsp* from your “Page” Component by right-clicking them – then select **Delete.**

2. Create a new JSP file named ***body.jsp*** in your “contentpage” Component.

Copyright 2010, Day Software AG, Switzerland

3. Cut the body code from *contentpage.jsp* and paste it in to *body.jsp* – then **Save**.

body.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<body>
    <h2>properties</h2>
    Title: <%= properties.get("jcr:title") %><br />

    <h2>currentPage</h2>
    Title: <%= currentPage.getTitle() %><br />
    Name: <%= currentPage.getName() %><br />
    Path: <%= currentPage.getPath() %><br />
    Depth: <%= currentPage.getDepth() %><br />

    <h2>currentNode</h2>
    Title: <%= currentNode.getProperty("jcr:title").getString() %><br />
    Name: <%= currentNode.getName() %><br />
    Path: <%= currentNode.getPath() %><br />
    Depth: <%= currentNode.getDepth() %><br />
</body>
```

NOTE

Notice the fact the ***global.jsp*** file has been included at the top of this script. Since this script will be included at runtime, it will be unaware of any previous includes of the global.jsp.

4. Open the file contentpage.jsp, and enter some JSP and HTML code, similar to below, replacing the <body> section – then Save.

contentpage.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <head>
        <title><%= currentPage.getTitle() == null ? currentPage.getName() :
currentPage.getTitle() %></title>
    </head>

    <cq:include script="body.jsp" />

</html>
```

5. Test your script by requesting a Page in CQ5 Siteadmin that implements this "Page" Component.

Copyright 2010, Day Software AG, Switzerland

- If successful, you should see no difference between what is displayed now, and what was displayed before.

Congratulations! You have successfully broken out a portion of your "Page" Component, and have included a script at runtime. Again, this type of technique will allow you to reuse scripts for different Components in the future.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Initialize the WCM

Goal

The following instructions explain how to initialize CQ5 WCM, enabling use of the Sidekick and other content management tools. These tools will allow content authors to work more efficiently. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

Why initialize the WCM?

The WCM initialization script performs all necessary steps to provide the whole WCM functionality on a page, including: Dialogs, Widgets, WCM CSS & JS, Sidekick, etc. Some WCM functionality such as the Page properties (no Dialog defined yet) may not work when only the initialization script is included. There are steps missing which we perform in a later exercise.

How to initialize the WCM in your "Page" Component:

1. Open the file **contentpage.jsp**, and enter some JSP code, similar to below, adding to the <head> section an include of the initialization script – then **Save.**

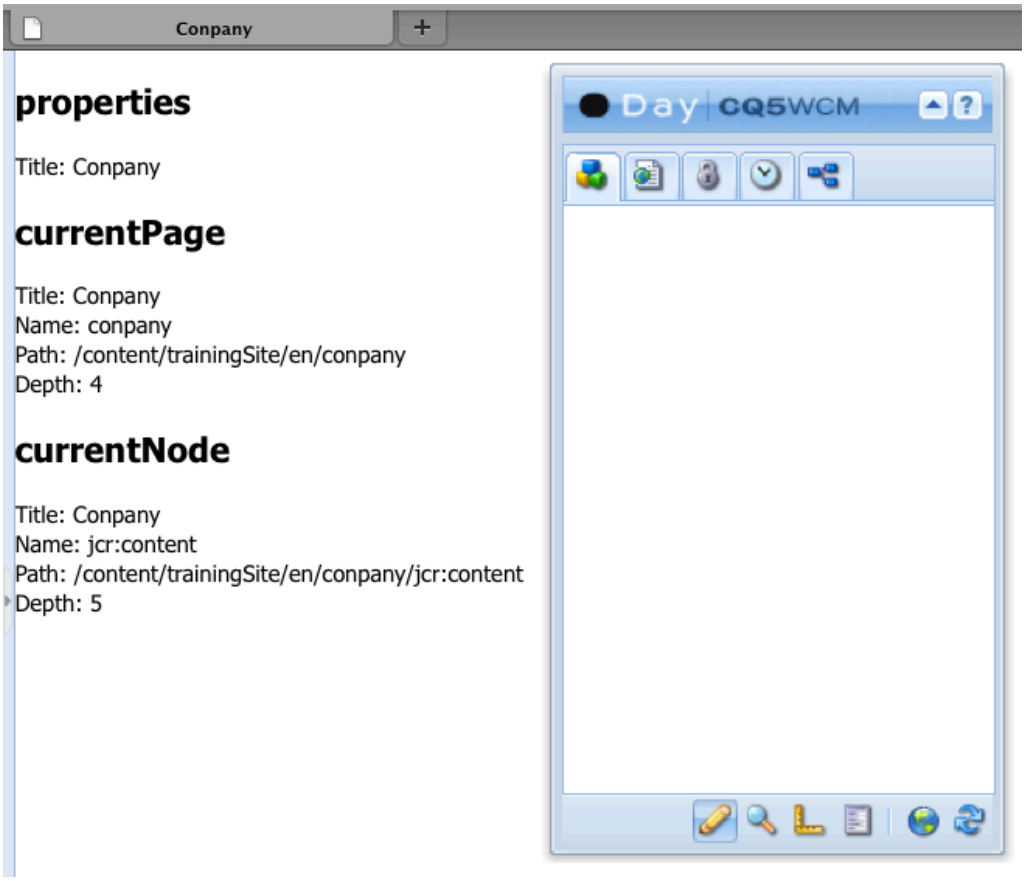
contentpage.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <head>
        <title><%= currentPage.getTitle() == null ? currentPage.getName() :
currentPage.getTitle() %></title>
    <cq:include script="/libs/wcm/core/components/init/init.jsp"/>
    </head>

    <cq:include script="body.jsp" />

</html>
```


Copyright 2010, Day Software AG, Switzerland

2. Test your script by requesting a Page in CQ5 Siteadmin that implements this "Page" Component.

- If successful, you should now see the Sidekick appear, which will give authors the ability to manage content.



Page view with WCM initialization

NOTE

For developers of previous CQ5 versions, you will notice that the location of the ***init.jsp*** has moved from */libs/wcm/init/init.jsp* to its current location.

Congratulations! You have successfully initialized the WCM, and all of its related tools. Again, this initialization is critical in enabling numerous CQ5 tools, allowing content authors to work more efficiently.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Extend the Foundation "Page" Component

Goal

The following instructions explain how to extend an existing foundation component, a Day recommended best approach, inheriting its resources thus allowing for script/Dialog reuse. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes

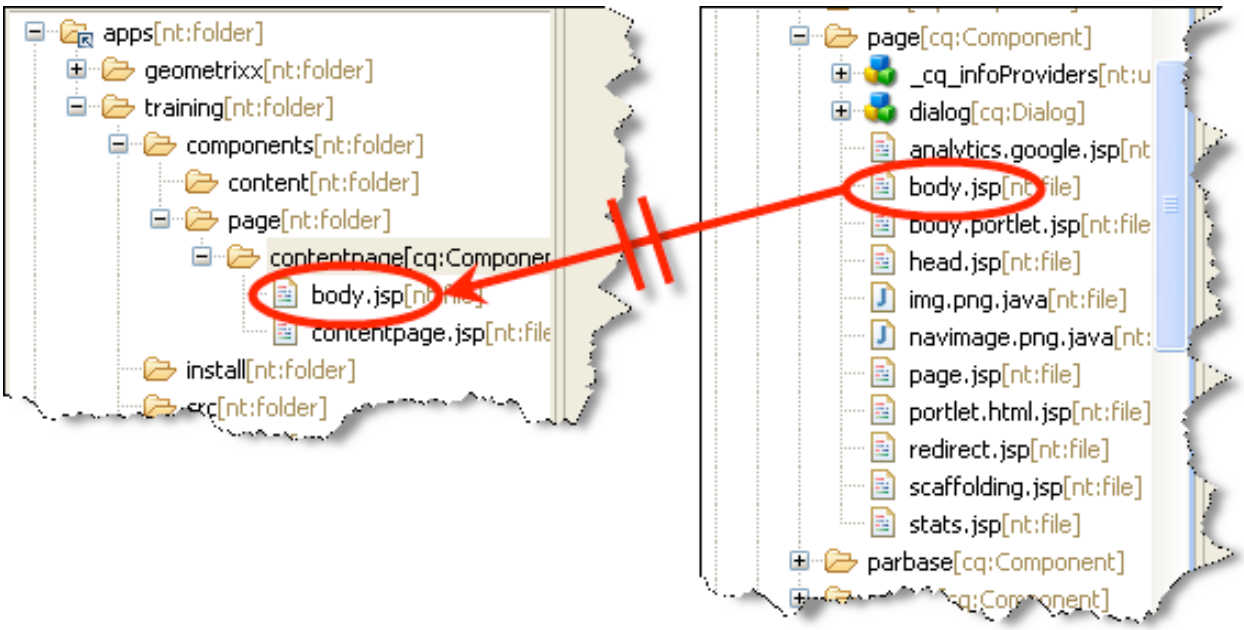
Why extend custom/foundation Components?

Components can be given a hierarchical structure to implement the inheritance of included script files, Dialogs, etc. Therefore it is possible for a specific “Page” Component (or any Component) to inherit from a “base” Component. For example, allowing inheritance of a script file for a specific part of the page (e.g. the <head> section).

Components within CQ5 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 from a “base” Component. For example, a text Component will inherit various attributes from the foundation text component, including:
    - scripts (resolved by Sling)
    - Dialogs
    - descriptions (including thumbnail images, icons, etc.)
  - Important to note is the fact that a local copy/instance of a Component element (e.g. body.jsp) will take precedence over an inherited element.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



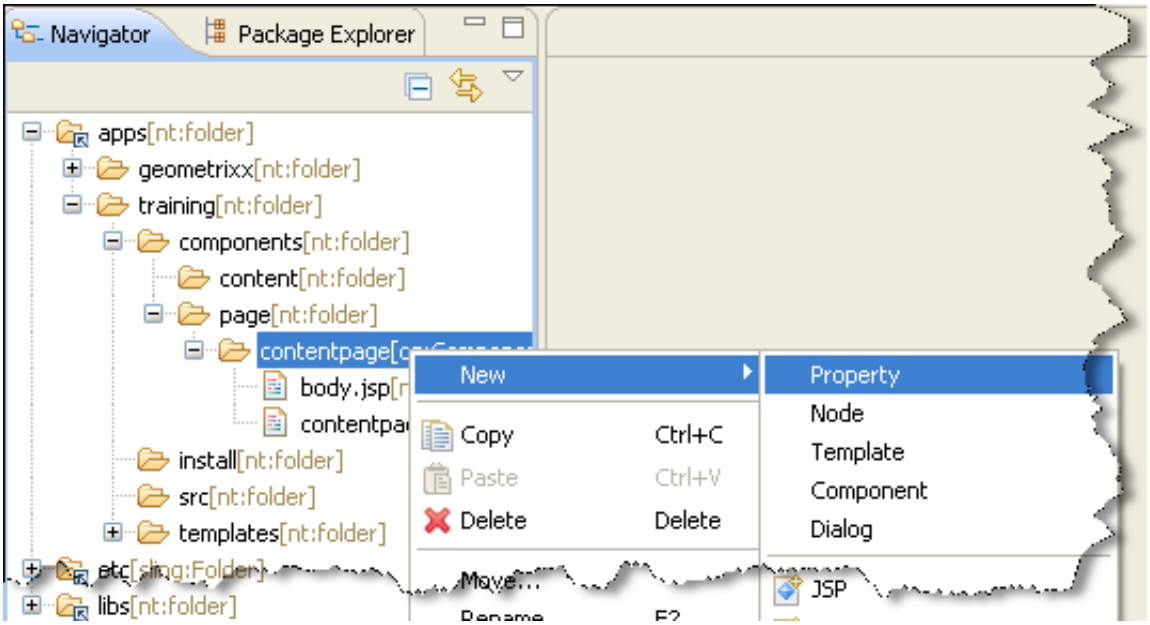
Non-inheritance of local copy

- Container Hierarchy
  - This is used to populate configuration settings to the child component and is most commonly used in a paragraph system scenario. For example, configuration settings for the edit bar buttons, control set layout (editbars, rollover, etc.), Dialog layout (inline, floating, etc.) can be defined on the parent Component and propagated to the children 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 typically 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 paragraph system, etc.

Copyright 2010, Day Software AG, Switzerland

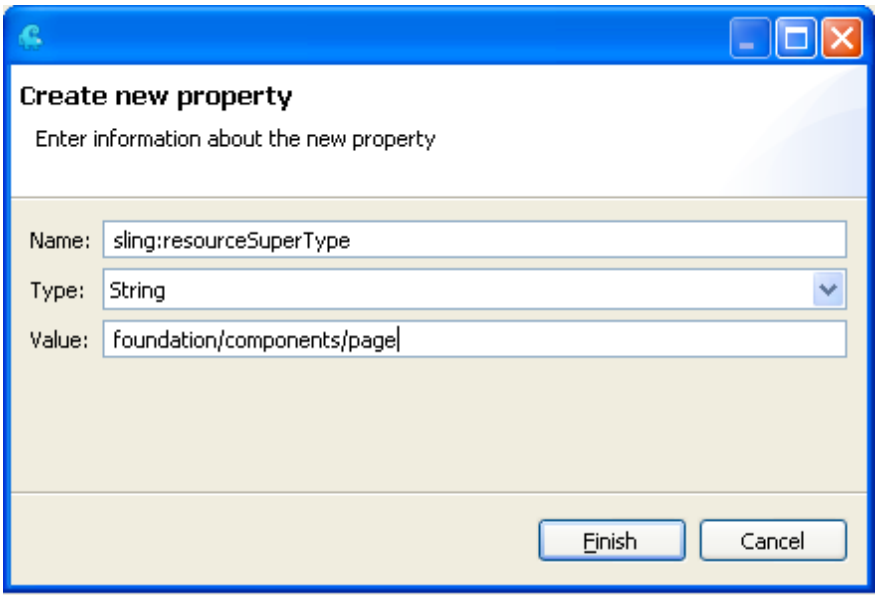
How to extend the foundation "Page" Component using CRXDE:

1. Right-click the Training contentpage component – then select **New, Property**.



CRXDE new property selection

2. Enter the property "Name" (***sling:resourceSuperType***), "Type" (***String***), and "Value" (***foundation/components/page***) in the dialog – then click **Finish**.

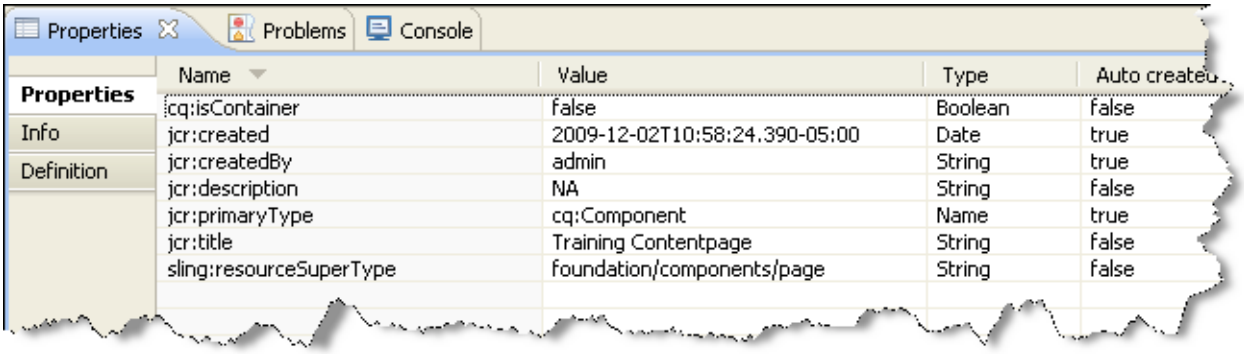


CRXDE new property dialog

Copyright 2010, Day Software AG, Switzerland

NOTE

You can view this property, and others directly related to the Training "Page" Component, by simply selecting and refreshing the "Properties" tab located at the bottom of CRXDE.



CRXDE properties view

3. Open the file **contentpage.jsp**, and enter some JSP code, similar to below, removing the <head> section and including the head.jsp script that we inherited from the foundation "Page" Component – then **Save**.

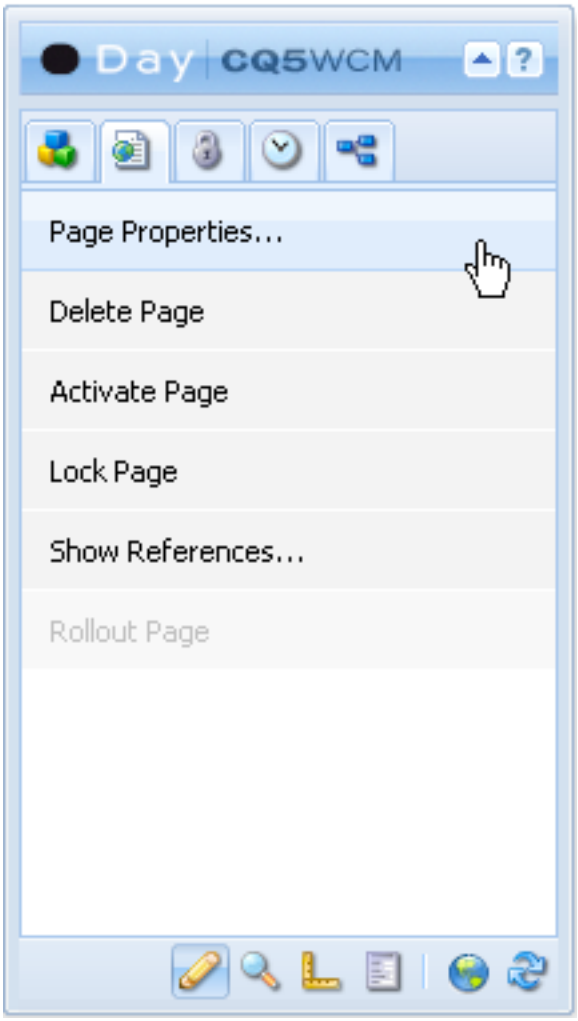
contentpage.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <cq:include script="head.jsp" />

    <cq:include script="body.jsp" />
</html>
```

4. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training "Page" Component.

- If successful, you should see no difference between what is displayed now, and what was displayed before.
- You can also see that the "Page Properties" functionality is now available in your Sidekick, since we inherited the Dialog from the foundation "Page" Component.

Copyright 2010, Day Software AG, Switzerland



Sidekick page properties selection

Congratulations! You have successfully extended an existing component, promoting reuse in your development efforts. Again, this type of technique will allow you to easily reuse scripts, Dialogs, etc. from Components that you may develop in the future.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Extend the Script Structure of the "Page" Component

Goal

The following instructions explain how to add additional structure to the existing Training “Page” Component, again allowing for resource reuse. This is merely an extension of a previous exercise where you added the <body> section of the “Page” Component to a new script (body.jsp), and then included that script. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes

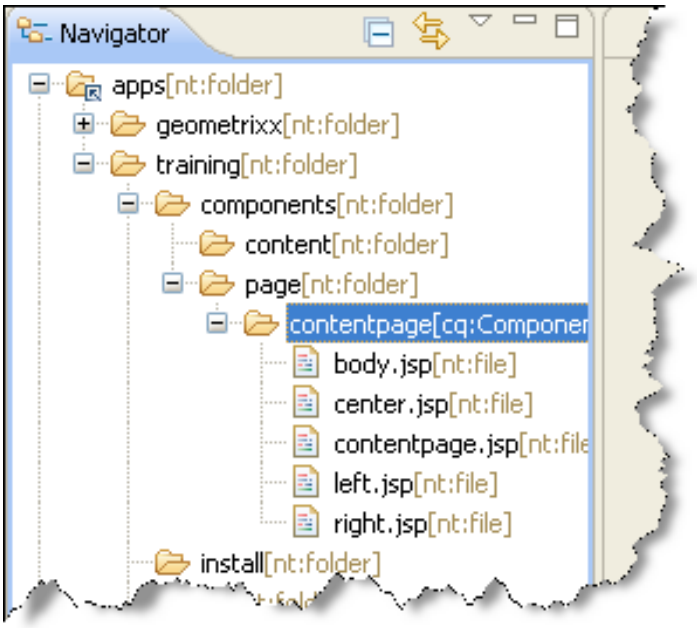
Why add additional structure?

Again, this exercise is merely an extension of a previous exercise where you added the <body> section of the “Page” Component to a new script (body.jsp), and then included that script. Day feels it is important you become well versed in this capability, as it will allow you to more easily reuse Components in the future. In addition, you are attempting to reflect the Geometrixx structure of its Contentpage Component, so that you may recreate the Web site in its entirety.

How to add additional structure to the “Page” Component:

1. Create 3 new JSPs (***left.jsp, center.jsp, right.jsp***) in the Training Contentpage “Page” Component.

Copyright 2010, Day Software AG, Switzerland



CRXDE newly created JSPs (left.jsp, center.jsp, right.jsp)

2. Open the file left.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

left.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="left">
    <div>logo</div>
    <div>newslist</div>
    <div>search</div>
</div>
```

3. Open the file center.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

center.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="center">
    <div>trail</div>
    <div>title</div>
    <div>par</div>
</div>
```


Copyright 2010, Day Software AG, Switzerland

4. Open the file right.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

right.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="right">
    <div>rightpar</div>
</div>
```

5. Open the file *body.jsp*, and enter some HTML and JSP code, similar to below – then Save.

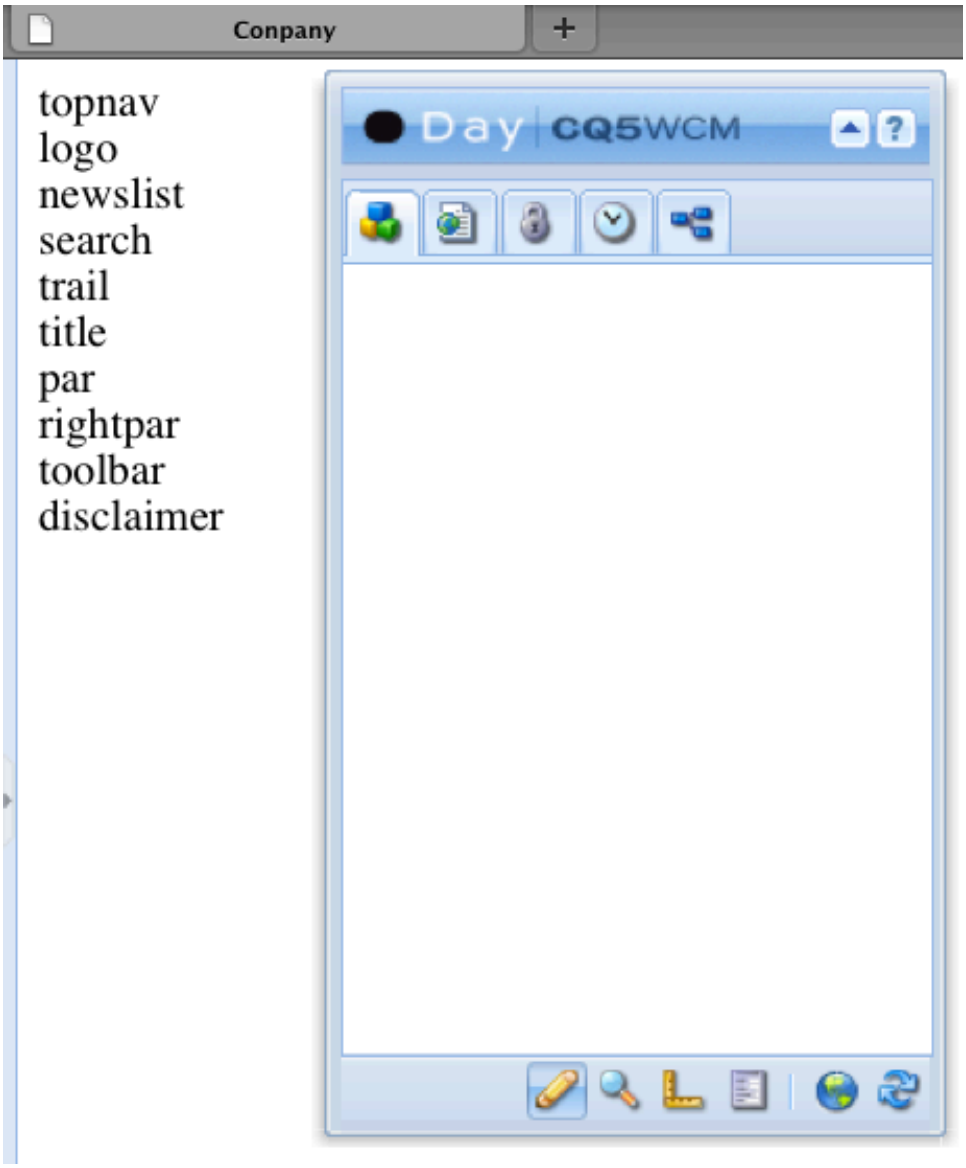
body.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<body>
    <div class="topnav">topnav</div>
    <div class="content">
        <cq:include script="left.jsp" />
        <cq:include script="center.jsp" />
        <cq:include script="right.jsp" />
    </div>
    <div class="footer">
        <div class="toolbar">toolbar</div>
        <div class="disclaimer">disclaimer</div>
    </div>
</body>
```

6. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training "Page" Component.

- If successful, you should see a simple text output (which we will fix soon) of words and the Sidekick, similar to the image below.

Copyright 2010, Day Software AG, Switzerland



Page view of additional structure

Congratulations! You have added additional structure to an existing Component, promoting reuse in your development efforts. Again, Day feels it is important you become well versed in this capability/technique, as it will allow you to more easily reuse Components in the future. Now let's make the output more attractive by introducing you to Designer functionality.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create and Assign a New Design

Goal

The following instructions explain how to create and assign a Design(er) to a Web site structure, allowing you to separate content and design, while promoting a consistent look and feel across many Pages. In the interest of time, we will create a new Design(er) for Training, but copy the existing static.css file from the Geometrixx Design(er). To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- Pages in which to assign the design

What is a Design(er) and why do I need it?

By creating and assigning a Design(er) in CQ5, it allows you to enforce a consistent look and feel across your Web site, as well as share global content. The many Pages that use the same Design(er), will have access to common CSS files, defining the formats of specific areas/Components, and images that you use for features, such as backgrounds and buttons.

CQ5 has been developed to maximize compliance with the Web Accessibility Guidelines. Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the Web, and that they can contribute to the Web.

This can include measures such as providing textual alternatives to images (or any non-text item). These can then be used to help people with sight impairment by outputting the text on a Braille keypad, or through a voice synthesizer. Such measures can also benefit people with slow internet connections, or any internet user - when the measures offer the user more information.

Such mechanisms must be carefully planned and designed to ensure that they provide the information required for the user to successfully understand and

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

use the content. Certain aspects are integral to CQ5, whereas other aspects must be realized during your project development.

How to create a Design(er) for your Web site, using CQ5 Siteadmin:

1. Navigate to the "**Tools**" section of CQ5.

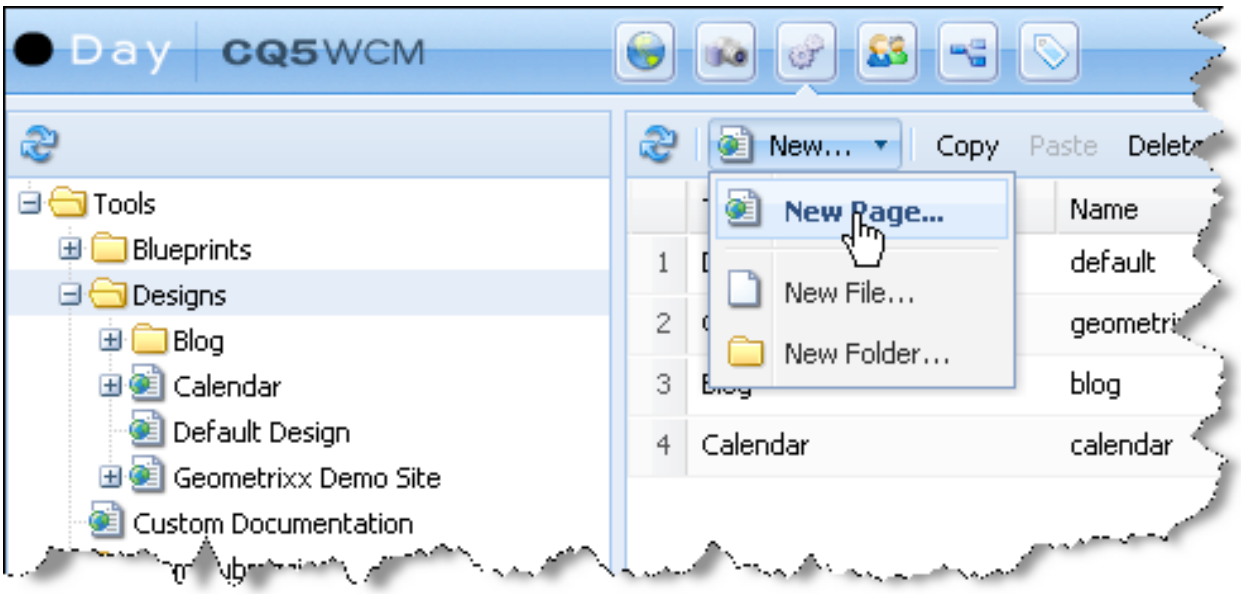
- URL: http://localhost:4502/miscadmin

CQ5 welcome page tools selection

CQ5 siteadmin tools selection

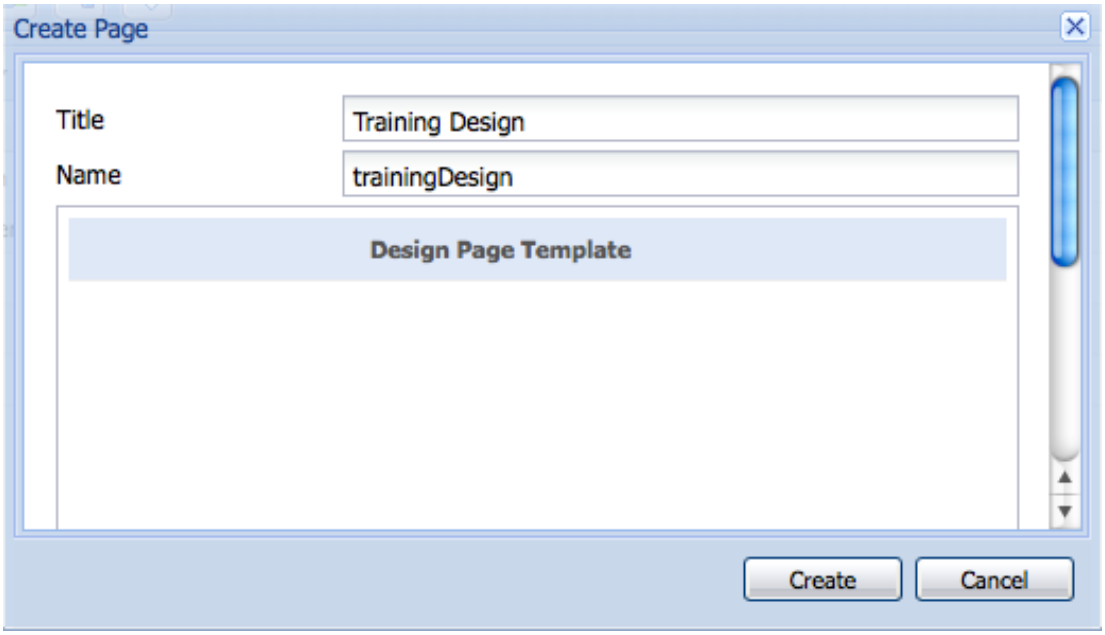
Copyright 2010, Day Software AG, Switzerland

2. Select the “Designs” folder – then select **New..., New Page ...**



Tools new page (design) selection

3. Enter the Design(er) “Title” (Training Design) and “Name” (trainingDesign) in the dialog – then select **Create.**



Tools create page (design)

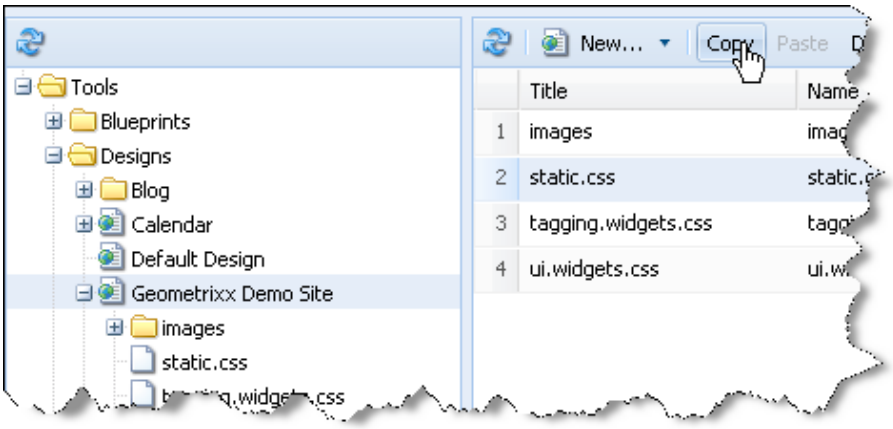
Copyright 2010, Day Software AG, Switzerland

NOTE

If you believe your implementation will have more than one Design(er), which is quite common, it is recommended you create a design structure that will allow multiple Design(er)s to be associated with the one project.

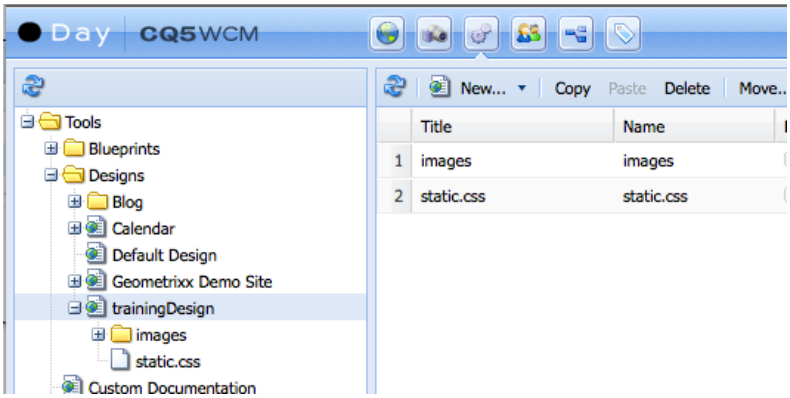
- /etc/designs/trainingDesign
  - /etc/designs/trainingDesign/default
  - /etc/designs/trainingDesign/products

4. Select the "Geometrixx" Design(er), afterward selecting the static.css and the images folder- then select **Copy.**



Tools copy static.css and images folder

5. Select the Training Design(er) – then select **Paste.**



Tools paste static.css

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully created a Training Design(er). You can easily modify the static.css file by using CRXDE and going to /etc/designs/training. Now that we have a Design(er), the next task is to assign that Design(er) to our existing Training Web site structure.

How to assign a Design(er) to your Web site, using CQ5 Siteadmin:

1. Navigate to the "**Websites**" section of CQ5.

- URL: http://localhost:4502/siteadmin

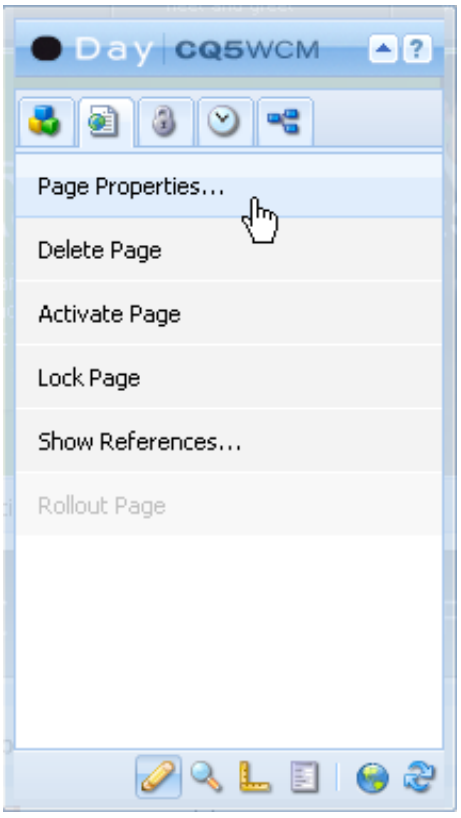
CQ5 welcome page websites selection

CQ5 tools siteadmin selection

2. Open the root Training Page (*/content/trainingSite*).

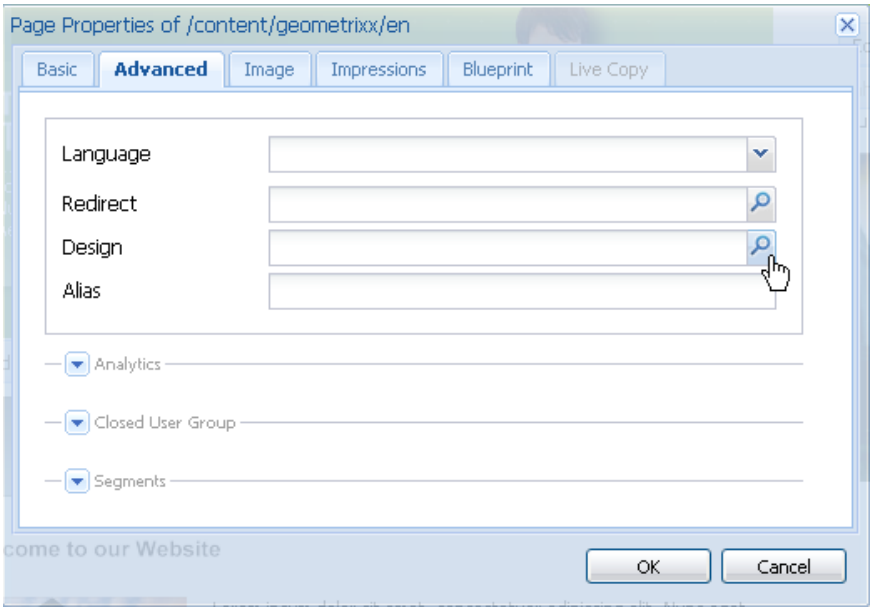
3. Select "**Page Properties**" in the Sidekick – a dialog will pop-up.

Copyright 2010, Day Software AG, Switzerland



Sidekick page properties selection

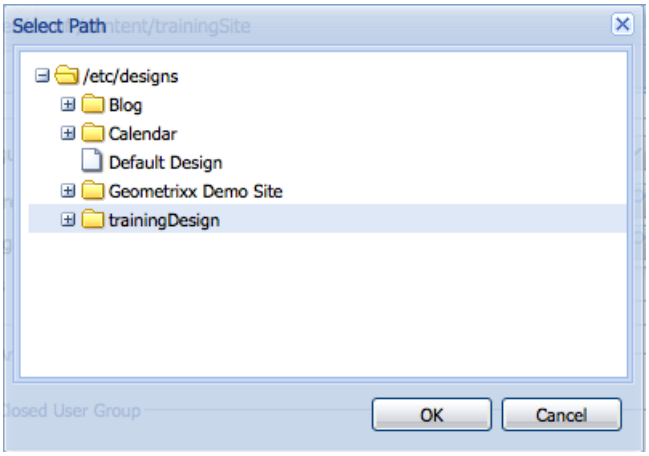
4. Select the "**Advanced**" tab – then select the "**Design**" option.



Page properties dialog - design selection

Copyright 2010, Day Software AG, Switzerland

5. Select the newly created **TrainingDesign** Design(er) – then select **OK, OK.**



Design selection dialog

Congratulations! You have successfully assigned a Training Design(er) to your Training Web site. You should now be able to view the "enhanced" Page, which ought to look similar to the image below. You will also notice the Sidekick no longer displays Components. As a Designer, you have not yet declared what Components can be used with this Design(er). Again, you can easily modify the static.css file by using CRXDE and going to /etc/designs/training.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create a Text-based Navigation Component

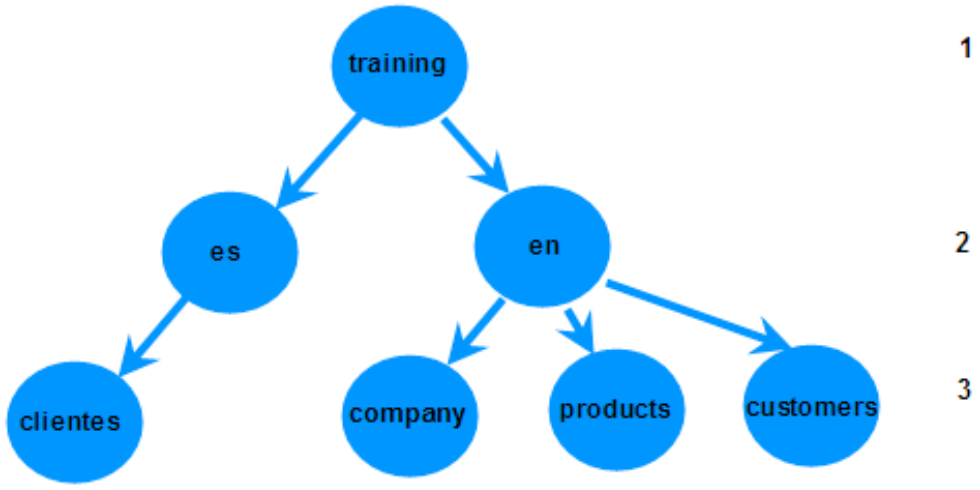
Goal

The following instructions explain how to create a dynamic text-based navigation Component, allowing for Web site structure to be easily modified and represented/navigated in real-time. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes
- Pages and Web site structure at least 4 levels deep (e.g. /content/trainingSite/en/company)

How do I create dynamic navigation?

Providing dynamic navigation capabilities, allowing for the easy addition and removal of Pages, is one of the most important (and sometimes difficult) things you can do as a developer in CQ5. Consider the following image, which represents a simple Web site structure:



Training web site structure

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

If you want to create a dynamic navigation Component that displays all the valid, children Pages of a language Page (e.g. “en”), you would need to identify the following:

- What Page is being requested?
  - Is it under the “en” section or the “es” section, as you will want to display the appropriate children of the language the visitor is browsing?
- At what level does the requested Page exist?
  - This is important because if you wish to display the children of a language, then a request to a level 1 Page (i.e. “training”) would break the assumption requests would only be made to level 2 (e.g. “en”) or level 3 (e.g. “company”) Pages.
- Is the requested Page valid?
  - In CQ5, it is possible to configure a Page to not display in any dynamic navigation script, as well as define a specific “On Time” and “Off Time”.

After these questions have been answered, it is relatively easy to collect the title of the Page, which will be displayed, and the path of the Page, which will be used to provide navigation functionality. Important Java classes/interfaces you will be using include:

- com.day.cq.wcm.api.Page
- com.day.cq.wcm.api.PageFilter

How to create a dynamic navigation Component:

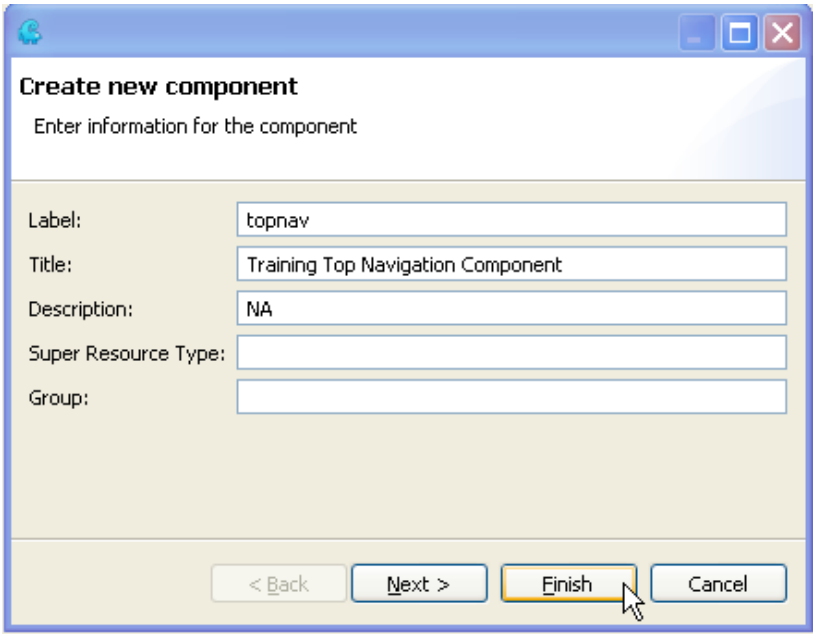
1. Right-click /apps/<application name>/components/content – then select **New, Component.**

2. Enter the desired Component “Label”, “Title”, “Description” – then click **Finish.**

- Label = the name of the Component/node that will be created
- Title (property jcr:title) = the title that will be assigned to the Component
- Description (property jcr:description) = the description that will be assigned to the Component
- Super Resource Type (property sling:resourceSuperType) = the parent Component in which this Component will inherit from

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- Group (property componentGroup) = the group in which this Component will be associated with when displayed in the Sidekick



CRXDE create component dialog

NOTE
This is a similar process to the "Page" Component we created earlier using CRXDE Lite.

3. Open the file ***topnav.jsp***, and enter some HTML and JSP code, similar to below – then **Save.**

topnav.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="java.util.Iterator,
                 com.day.cq.wcm.api.PageFilter" %>
<%
// get navigation root page
Page navRootPage = currentPage.getAbsoluteParent(2);

// check to make sure the page exists
if (navRootPage == null) {
      navRootPage = currentPage;
}

if (navRootPage != null) {
      // create an iterator object of all nav root's child pages
      Iterator<Page> children = navRootPage.listChildren(new PageFilter());
```


Copyright 2010, Day Software AG, Switzerland

```
      while (children.hasNext()) {
            // get next child page
            Page child = children.next();
            // display the link in an <A HREF...
            %><a href="<%= child.getPath() %>.html"><%=child.getTitle() %></
a><br /><%
      }
}
%>
```

4. Open the file ***body.jsp*** in the Training Contentpage Component and replace the "topnav" section with a <cq:include> of the navigation Component you just created, similar to below – then **Save**.

- 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.
- 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.

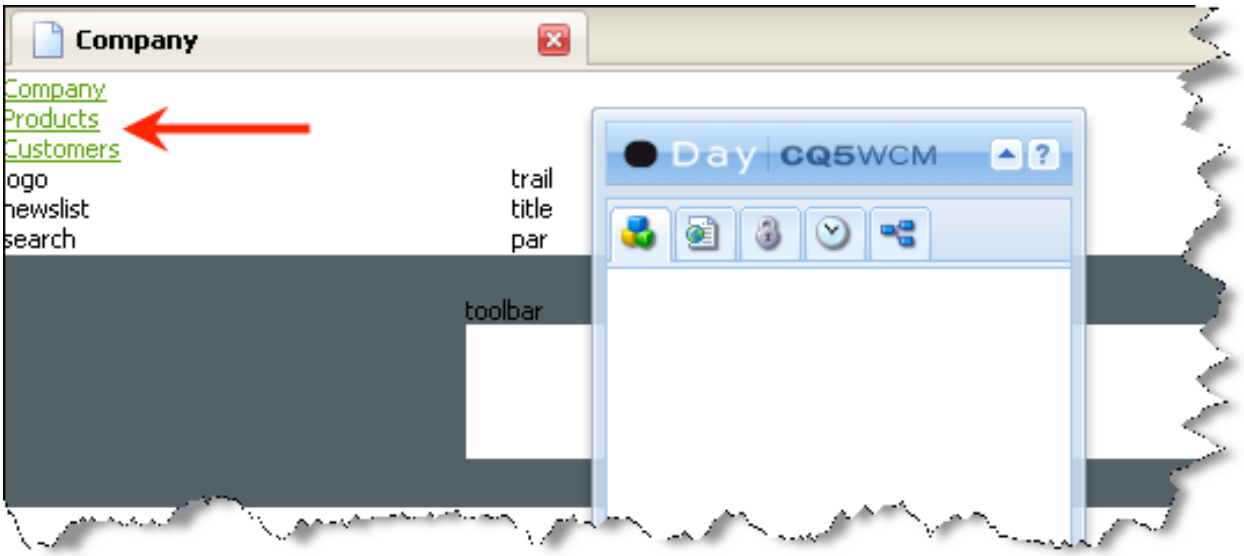
body.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<body>
    <cq:include path="topnav" resourceType="training/components/content/
topnav"/>
    <div class="content">
        <cq:include script="left.jsp" />
        <cq:include script="center.jsp" />
        <cq:include script="right.jsp" />
    </div>
    <div class="footer">
        <div class="toolbar">toolbar</div>
        <div class="disclaimer">disclaimer</div>
     </div>
</body>
```


Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

5. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training “Page” Component.

- If successful, you should see navigation links/titles directly related to your Web site structure.



Page view with topnav

Congratulations! You have successfully created a “Content” Component that dynamically represents the navigation structure of your Web site. This is a perfect example of how you can easily create and integrate Components into your Templates.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Day

EXERCISE - Add a Log Message

Goal

The following instructions explain how to add a log message to a Component script. This will allow you to more easily debug various scripts you may be working on. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed navigation Component

Why add log messages?

In the daily life of a developer, it is often crucial to monitor the values of variables assigned/used. There are several possibilities, in various usability levels. CQ5 and CRXDE make your life a little easier by implementing the popular Log4j framework, which is designed to provide an easy-to-use logging solution.

The initialization of a "Logger" object, called "log", has already been accomplished during the inclusion of *global.jsp* in whatever Component you may be working on. The log file entries are formatted according to the Sling configuration. Two pieces of information are required to append an entry to the log file:

- log level
  - This is provided by the corresponding method call. For example, a log.debug(<message>) produces a message with log level "debug", while a log.info(<message>) produces a message with log level "info". Possible methods of the "Logger" object include:
    - trace()
    - debug()
    - info()
    - warn()
    - error()

Copyright 2010, Day Software AG, Switzerland

- message
  - The message itself is provided as a parameter to the method call. For example, log.debug("This is the log message") appends the message "This is the log message" with a log level of “debug” to the error.log file.

NOTE

In a default configuration, the log file is located under <cq-install-dir>/crx-quickstart/logs/error.log.

How to add a log message:

1. Open the file ***topnav.jsp*** in the Training “topnav” Component, and enter some JSP code, similar to below – then **Save**.

topnav.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="java.util.Iterator,
                 com.day.cq.wcm.api.PageFilter" %>
<%
// get navigation root page
Page navRootPage = currentPage.getAbsoluteParent(2);

// check to make sure the page exists
if (navRootPage == null) {
      navRootPage = currentPage;
}

if (navRootPage != null) {
      // create an iterator object of all nav root's child pages
      Iterator<Page> children = navRootPage.listChildren(new PageFilter());

      while (children.hasNext()) {
            // get next child page
            Page child = children.next();
            // new log message -- logging title of nav page
            log.info("child page [{}] found.", child.getTitle());
            // display the link in an <A HREF...
            %><a href="<%= child.getPath() %>.html"><%=child.getTitle() %></
a><br /><%
      }
}
%>
```


Copyright 2010, Day Software AG, Switzerland

2. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training "Page" Component.

- If successful, you should see no difference between what is displayed now, and what was displayed before. However, if you were to inspect the error.log file, you would see a message similar to below:

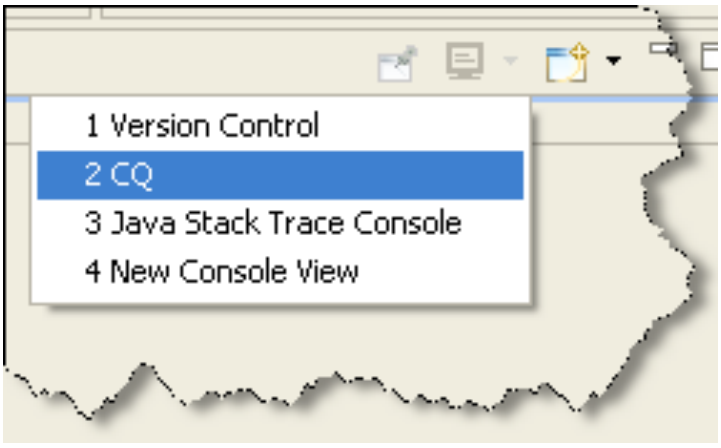
```
04.12.2009 11:23:22.921 *INFO* [127.0.0.1 [1259943802812] GET /content/
trainingSite/en/company.html HTTP/1.1]
apps.training.components.content.topnav.topnav$jsp child page [Company] found.
```

Congratulations! You have successfully added a log message to your script. Again, it is often crucial to monitor the values of variables assigned/used. The "Logger" object is a useful tool that will enable you to do so.

NOTE

In CRXDE you can display the file error.log that is located on the file system at <cq-install-dir>/crx-quickstart/server/logs and filter it with the appropriate log level. This can be done by proceeding as follows in CRXDE:

1. Select the **Console** tab located at the bottom of the window – then select the arrow beside the **Open Console** – then select **System Log**.

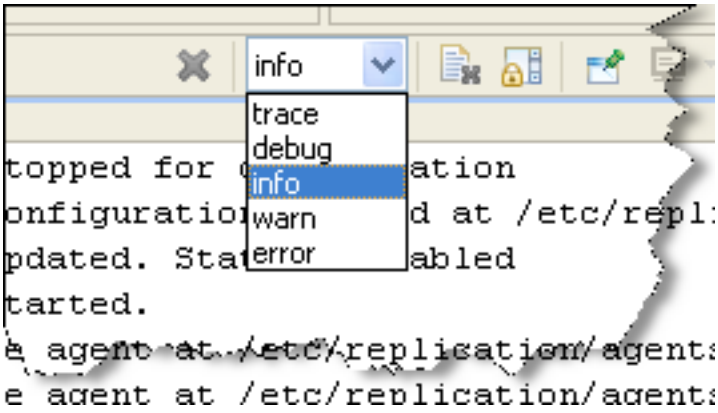


CRXDE console application log select

Copyright 2010, Day Software AG, Switzerland

2. Select the desired “log level” in the drop down menu.

- The error.log is then filtered according to the log level and displayed in the tab.



CRXDE console log level select

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Enable the Debugger

Goal

The following instructions explain how to enable the debugger in CQ and CRXDE. This will allow you to more easily debug various scripts you may be working on. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed navigation Component

Why enable the debugger?

Sometimes, outputting variable values in a log file is not enough for a developer to monitor the sequence of steps executed by the system. The best way to accomplish this is the usage of a debugger. Many modern IDEs like Eclipse and CRXDE provide this functionality.

Again, the goal of this exercise is to learn how to use the built-in debugger in CRXDE. A debugging session consists of a server and a client. In this training, you will use CRXDE as the client, while the server is provided by CQ. For performance reasons, Day recommends not enabling debugging for CQ production instances, as this would consume too many resources, thus hindering performance.

How to enable debugging in CQ and CRXDE:

1. Shutdown your CQ author and CRXDE instance.

2. Navigate to the directory where the Java servlet engine (CQSE) has been installed using your command line.

- e.g. <cq-install-dir>/crx-quickstart/server

3. Enter the following command to start the CQ server in debug mode.

Copyright 2010, Day Software AG, Switzerland

- Windows = server.bat -debug socket
- Linux/Unix = start -d

DOS command line start/enable of debugger

NOTE

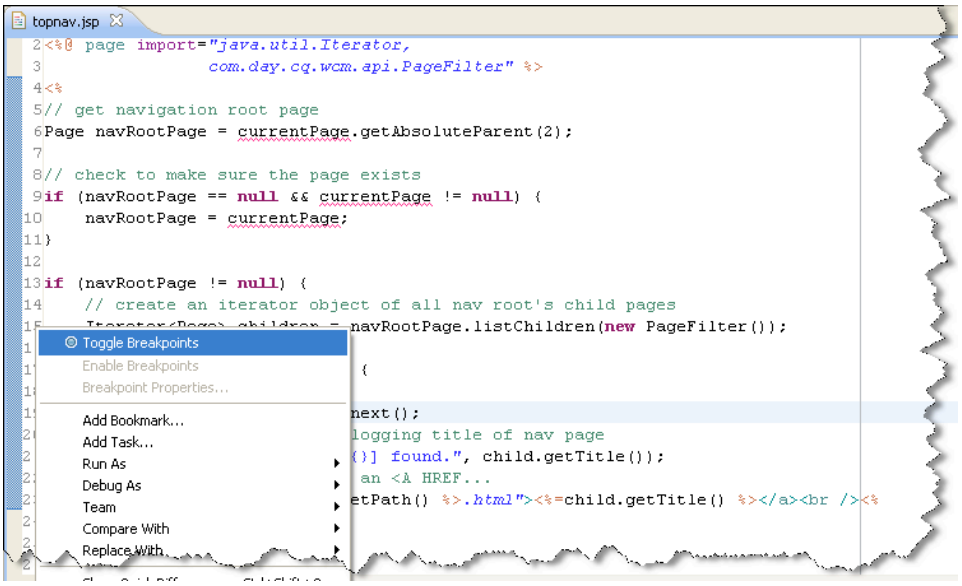
By default, port 30303 will be used by the server for debugging communication. You can customize the server.bat/start file for future use. In addition, you can call the debugger directly:

java -Xmx512m -Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=30303,suspend=n -jar cq-author-4502.jar

4. Start CRXDE.

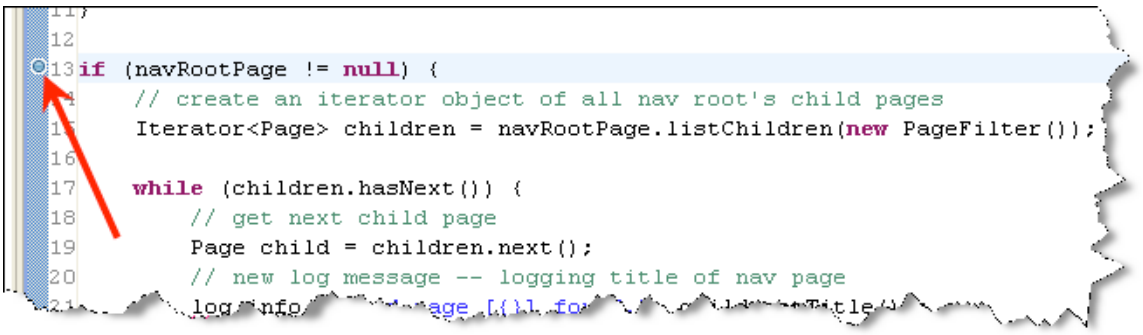
5. Open the file ***topnav.jsp*** in the Training "topnav" Component.

6. Right-click the left-side of the JSP editor on a line of code (e.g. line 13: if (navRootPage != null) {) - then select **Toggle Breakpoints.**



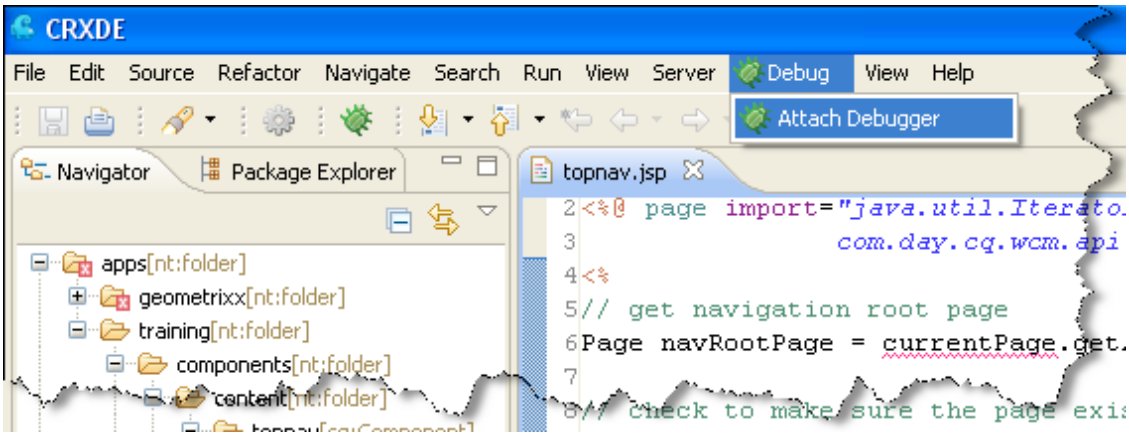
CRXDE right-click JSP editor

Copyright 2010, Day Software AG, Switzerland



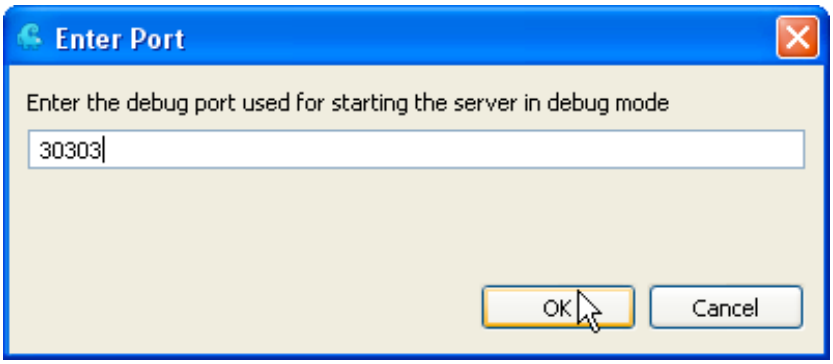
CRXDE code toggle breakpoint

7. Switch to debug mode by selecting **Debug, Attach Debugger**.



CRXDE switch to debug mode

8. Enter the debug port (30303) used for starting the server in debug mode – then select **OK**.

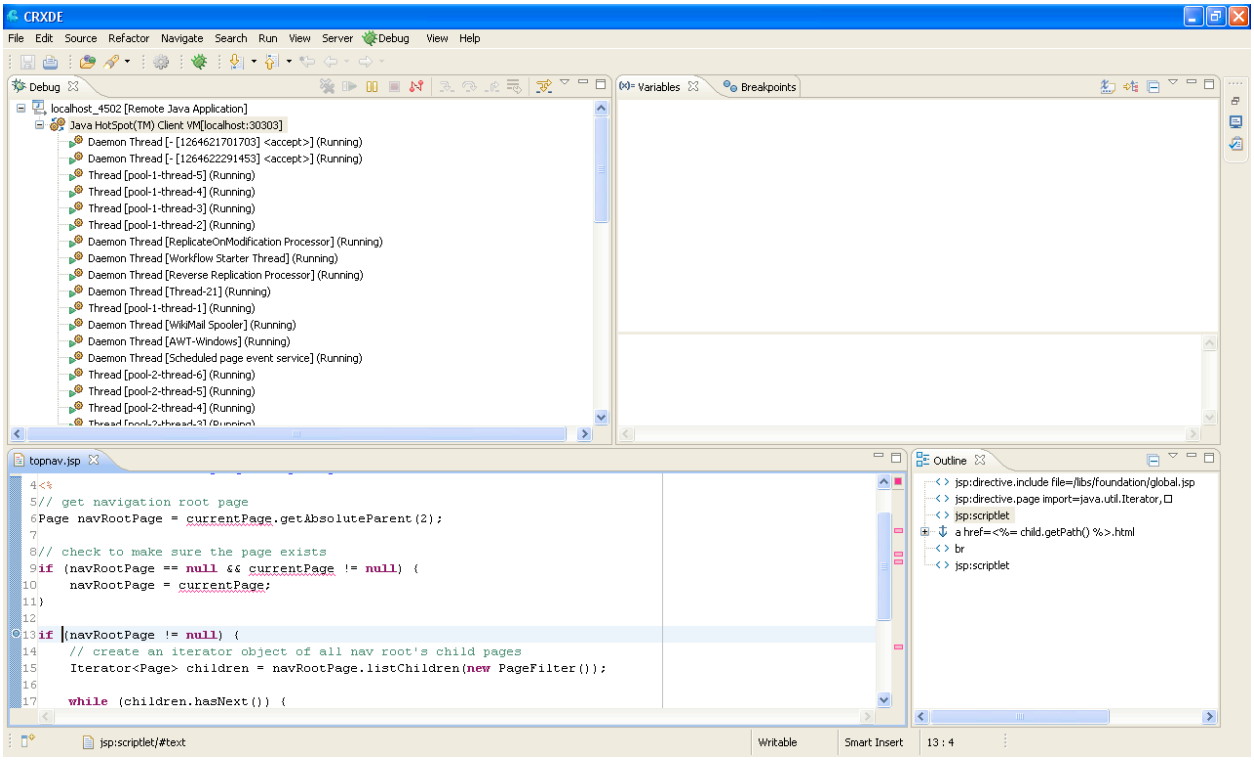


CRXDE debug port dialog

Copyright 2010, Day Software AG, Switzerland

NOTE

You are now in debug mode. Your CRXDE view should look similar to the image below.



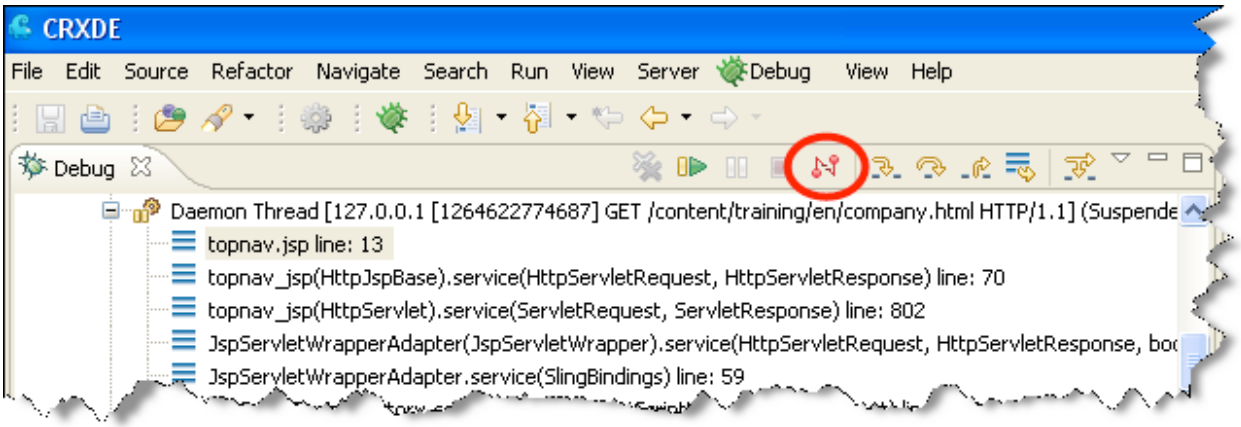
CRXDE debug mode

9. Test your script/debug mode by requesting a Page in CQ5 Siteadmin that implements this Training “topnav” Component.

- In CRXDE, you will notice your script stops at the line of code containing the breakpoint
- Here you will be able to investigate variables, execution, and other related elements, in addition to "stepping" through your code

10. Select the **Disconnect** button to disable debug mode.

Copyright 2010, Day Software AG, Switzerland



CRXDE disconnect debug mode

Congratulations! You have successfully enabled debugging in CQ and CRXDE. Again, this is a great tool that will allow you to monitor the sequence of steps executed by the system.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create an Image-based Navigation Component

Goal

The following instructions explain how to create a dynamic image-based navigation Component, allowing for Web site structure to be easily modified and represented/navigated in real-time. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes
- A completed text-based top navigation Component (e.g. /apps/training/components/content/topnav)
- Pages and Web site structure at least 4 levels deep (e.g. /content/trainingSite/en/company)

How do I transform images dynamically?

Once again, providing dynamic navigation capabilities, allowing for the easy addition and removal of Pages, is one of the most important (and sometimes difficult) things you can do as a developer in CQ5. In this exercise, you will modify the existing text-based navigation Component to output background images overlaid with text equal to the title of a Page you wish to allow navigable.

To be able to use image based navigation items, you need a mechanism to request a page in an "image navigation item view". To achieve this, you will add a selector to the request of the navigation item image of a page (e.g. /path/to/a/page.*navimage.png*). Requests with such a selector have to be handled by an image processing mechanism. To achieve this, you will use Sling's request processing mechanism.

You will need to add an image processing script or Java servlet that will handle all requests with the specific selector "**navimage.png*". For the rendering of images with an overlay of text, the abstract Java servlet AbstractImageServlet is

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

very helpful. You will merely have to overwrite the method createLayer() to implement your desired logic.

Relevant functionalities (API calls) you will use are:

- 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

In the interest of time, Day Training has provided the Java servlet and background image that will achieve this image processing/text overlay. You will simply perform a code review in class. These items can be found in: <usb>/exercises/image-navigation.

For more detailed information concerning the AbstractImageServlet and other Day related image processing capabilities, please review the CQ Javadoc provided with CQ WCM and on your USB.

How to create an image-based navigation Component:

1. Copy the files ***navimage.png.java*** and ***navimage_bg.jpg*** from the USB (<usb>/exercises/image-navigation) – then paste to the Training Contentpage Component.

- e.g. /apps/training/components/page/contentpage

CRXDE structure of contentpage component

Copyright 2010, Day Software AG, Switzerland

2. Review the ***navimage.png.java*** file for a better understanding of the AbstractImageServlet.

WARNING

It is possible you will need to modify the Java package to match the current location of the Training Contentpage Component.

- e.g. apps.training.components.page.contentpage

3. Open the file ***topnav.jsp*** in the Training top navigation Component (i.e. /apps/training/components/content/topnav), and enter some HTML and JSP code, similar to below – then **Save.**

topnav.jsp

```
<%@include file="/libs/wcm/global.jsp"%>
<%@ page import="java.util.Iterator,
                 com.day.cq.wcm.api.PageFilter" %>
<%
// get navigation root page
Page navRootPage = currentPage.getAbsoluteParent(2);

// check to make sure the page exists
if (navRootPage == null) {
    navRootPage = currentPage;
}

if (navRootPage != null) {
    // create an iterator object of all nav root's child pages
    Iterator<Page> children = navRootPage.listChildren(new PageFilter());

    while (children.hasNext()) {
        // get next child page
        Page child = children.next();
        // new log message -- logging title of nav page
        log.info("child page [{}] found.", child.getTitle());
        // display the link in an <A HREF...
        %><a href="<%= child.getPath() %>.html"><img alt="<%= child.getTitle
() %>" src="<%= child.getPath() %>.navimage.png"></img></a><%
    }
}
%>
```


Copyright 2010, Day Software AG, Switzerland

4. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training "Page" Component.

- If successful, you should see navigation links/titles directly related to your Web site structure.

Page preview of image navigation

Congratulations! You have successfully modified an existing "content" navigation Component that dynamically represents the navigation structure of your Web site using images.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create a Title Component

Goal

The following instructions explain how to create a Component that allows authors to manage a Page's title (textual content), through the use of a simple Dialog. This is a beginning step in learning how to create Components that allow authors to write content. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

What is a Dialog?

During this training you have focused mostly on rendering content (static and dynamic), which is important as one could argue it comprises 50% of what you do as a CQ5 developer. What needs to be accomplished for this exercise is to provide content authors the ability to write content. You can accomplish this by learning how to create a Component Dialog. A CQ5 Dialog is similar to other dialogs you have used/created in the past: it gathers user input via a "form", potentially validates it, and then makes that input available for further use (storage, configuration, etc.).

CQ5 makes use of the popular ExtJS JavaScript framework. It allows developers the ability to easily create Rich Internet Applications (RIA) through the use of AJAX. It includes:

- High performance, customizable UI widgets
- Well designed and extensible Component model
- An intuitive, easy to use API

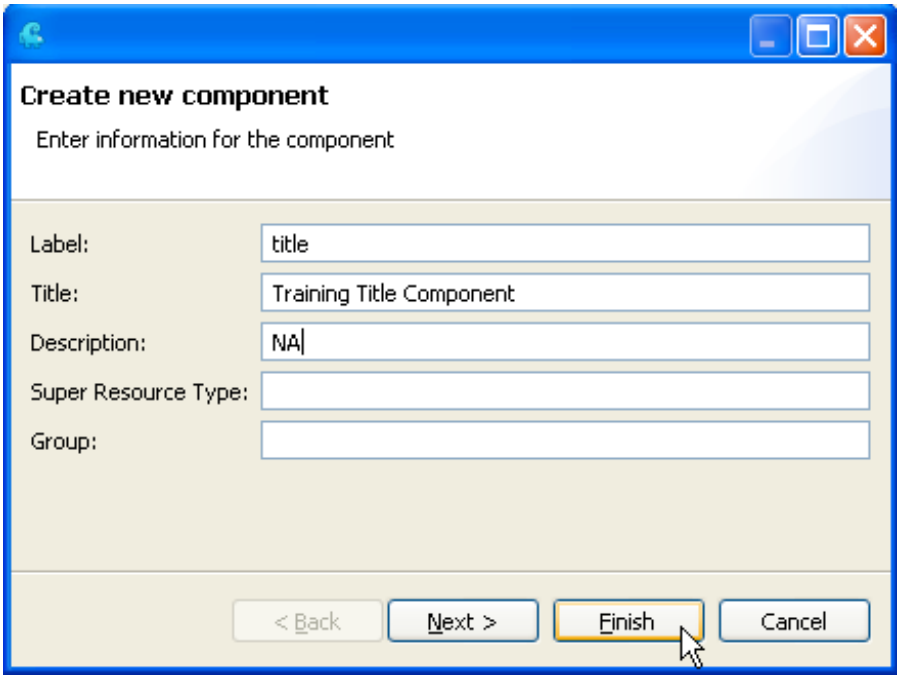
In relation to the Component the Dialog is being created for, the root node of the Dialog has to be of node type "cq:Dialog" and named "dialog". Below this Dialog root node, the nodes for the tabs of the Dialog have to be added. These

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

"tab nodes" must be of node type "cq:WidgetCollection". Below the "tab nodes" the "widget nodes" can then be added. The "widget nodes" must be of node type "cq:Widget".

How to create a title Component with a Dialog:

1. Create a new title "content" Component.



CRXDE new component dialog

NOTE

This process is similar to creating the navigation Component.

2. Open the file title.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

title.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<h1><%= properties.get("title", currentPage.getTitle()) %></h1>
```


Copyright 2010, Day Software AG, Switzerland

NOTE

You will notice the use of the Sling ValueMap class/object "properties". This object allows you to easily get properties associated with this instance of this Component.

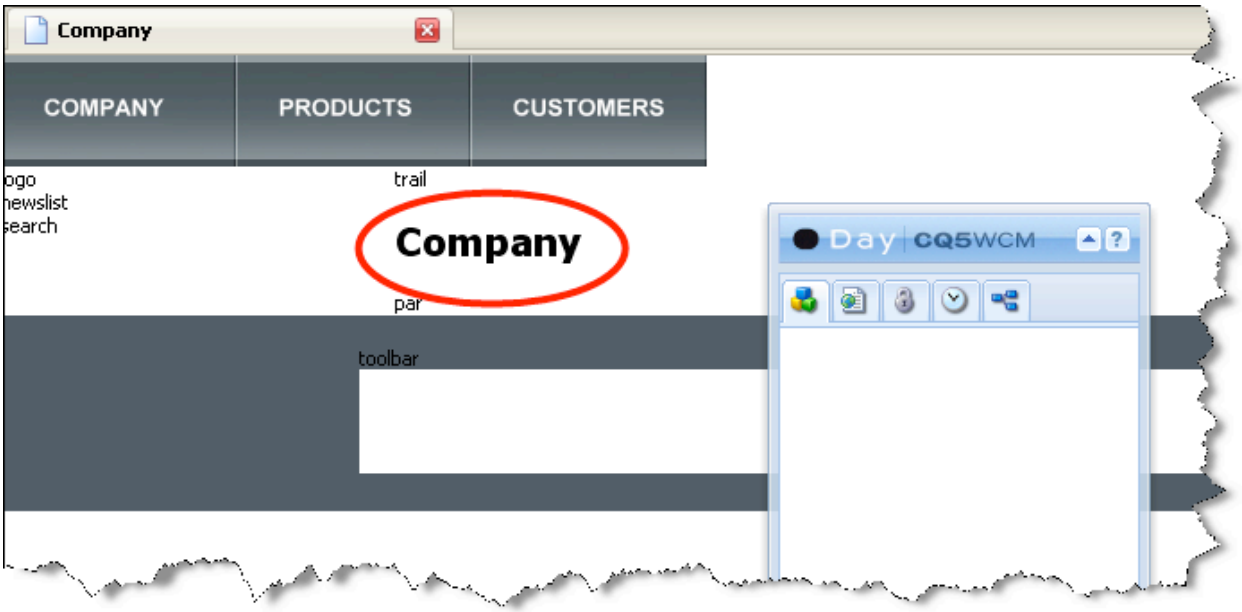
3. Open the file center.jsp in the Training Contentpage Component and replace the "title" section with a <cq:include> of the title Component you just created, similar to below – then **Save**.

center.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="center">
    <div>trail</div>
    <cq:include path="title_node" resourceType="training/components/content/
title" />
    <div>par</div>
</div>
```

4. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training title Component.

- If successful, you should see the default Page title, similar to the image below.

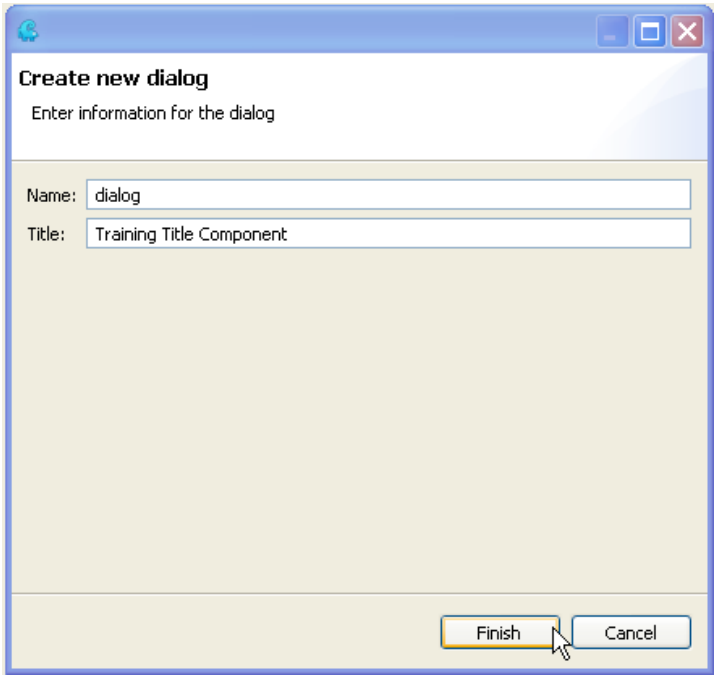


Page default view

5. Right-click the title Component – then select **New, Dialog**.

Copyright 2010, Day Software AG, Switzerland

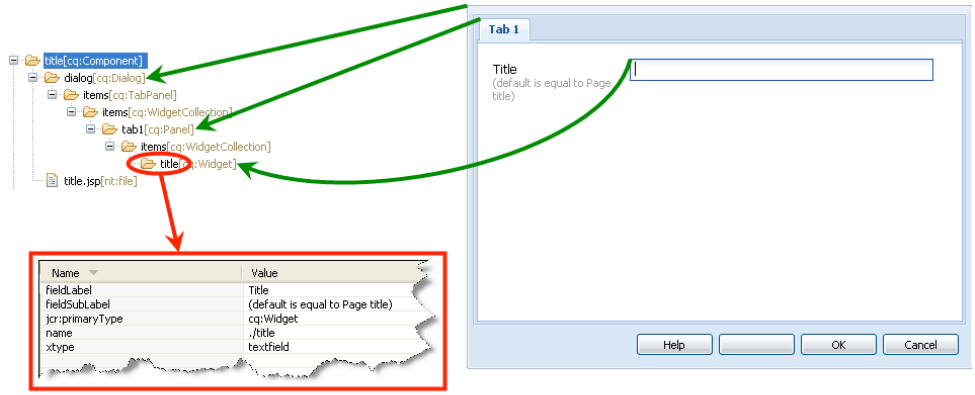
6. Enter the desired Dialog "Name" (dialog) and "Title" (Training Title Component).



CRDXE new dialog dialog

NOTE

Now is a good time for you to review the Dialog/node structure, and be able to easily map the related elements. Please review the image below for a better understanding (image is a completed Dialog - you are not there yet).

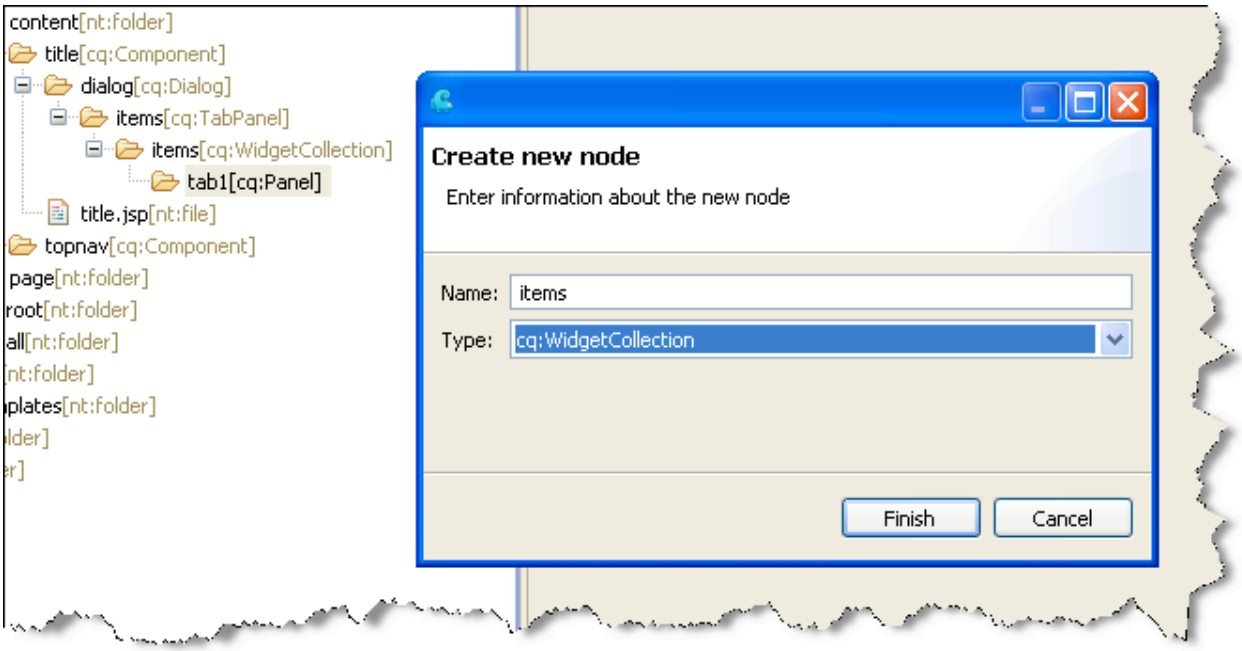


Dialog/node relational map

Copyright 2010, Day Software AG, Switzerland

7. Right-click the *tab1* node - then select **New, Node.**

8. Enter the desired "Name" (items) and "Type" (cq:WidgetCollection).



CRXDE new node dialog

9. Create a new node under the newly created *items* node.

- Name = title
- Type = cq:Widget

10. Right-click the newly created *title* node - then select **New, Property**.

11. Enter the desired "Name", "Type", and "Value" for the properties listed below.

- the property that will define where the content is stored
  - Name = name
  - Type = String
  - Value = ./title

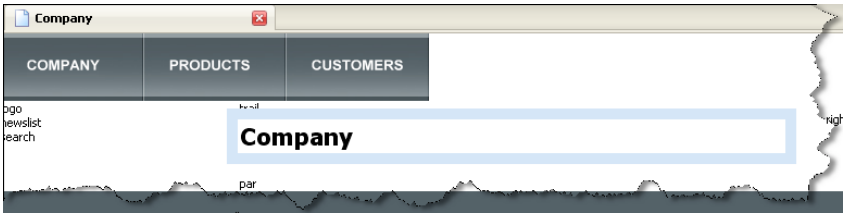
Copyright 2010, Day Software AG, Switzerland

- the property that will define the Widget type (i.e. user interface control)
  - Name = xtype
  - Type = String
  - Value = textfield
- the property that will define the label applied to the Widget
  - Name = fieldLabel
  - Type = String
  - Value = Enter Title Here

NOTE

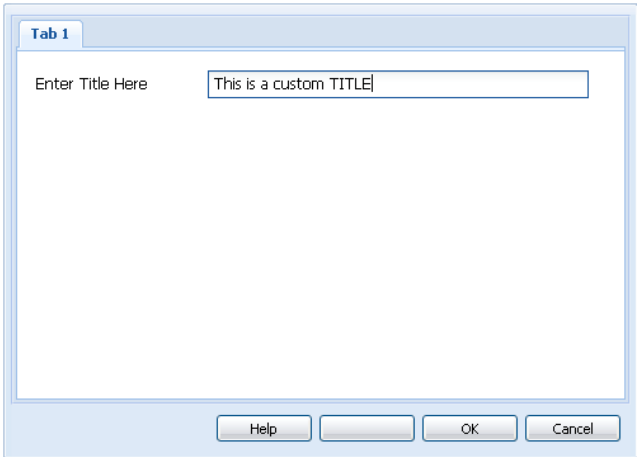
These are but a few of the many properties and Widgets you can use to create a Dialog that meets the needs of your users/content authors. You can find a complete listing in the provided CQ-WIDGETS-API on the USB.

12. Test your script/Dialog by requesting a Page in CQ5 Siteadmin that implements this Training title Component - then double-click the "title" content to invoke the Dialog.



Page title content selection

13. Enter the desired custom title - then select **OK**.



Dialog of title

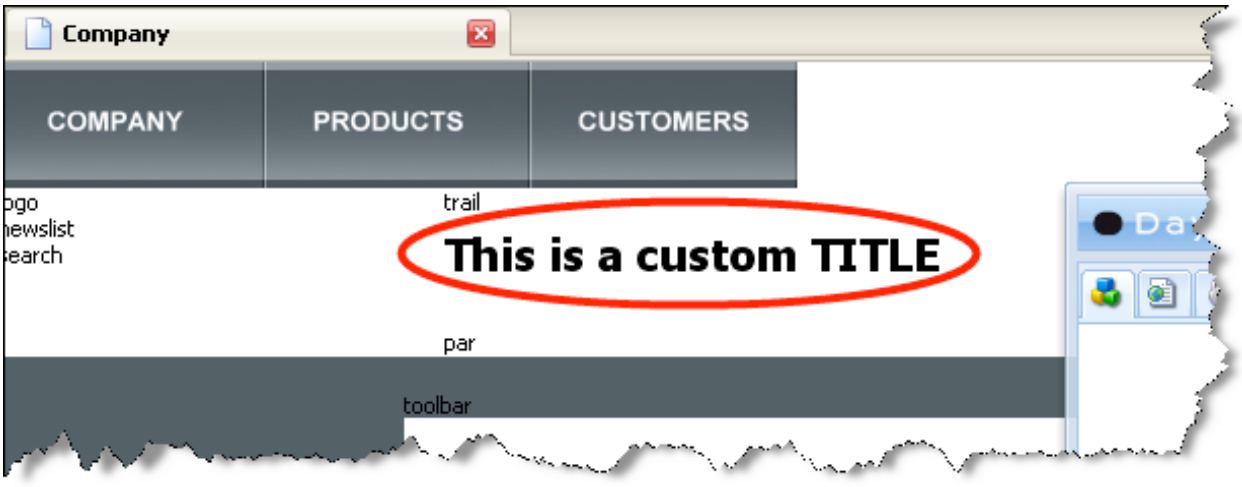
Copyright 2010, Day Software AG, Switzerland

NOTE

Notice how tab1's label is "Tab 1." If you want to change this value, simply assign a property named title (type String) to the tab1 node, and assign a value.

14. Review the Page output of the title content.

- If successful, you should see the custom Page title, similar to the image below.



Page view with custom title

Congratulations! You have successfully created a title Component, with a Dialog, that allows content authors to write content and have it displayed. This is a significant step in your development capabilities, as you are now able to create CQ Components that allow you to write content to the repository.

Copyright 2010, Day Software AG, Switzerland

Extra Credit EXERCISE - Create a List Children Component

Goal

The following exercise tests your knowledge of component creation and taking input from an author. The goal is to create a ListChildren component. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page-rendering" Component (e.g. Training Contentpage) with appropriate includes

Create a List Children Component

Using the Page, PageFilter, and PageManager classes, create a **listChildren** component. Remember you can look up these classes in the java docs. You have all the information you need, from the **topnav** and **title** components, to complete this component.

1. Include/import the appropriate java classes.

2. Get a property (should be a path) whose value will become the root of the list

3. If the list root property is empty, use a default. (static path or path to current page)

4. Use the path to get the root page.
(hint: the PageManager class has a method to get a page when you know its path)

5. Set up a page iterator, filtering by valid pages

Copyright 2010, Day Software AG, Switzerland

6. Iterate over the children

7. For each child, write out a link

8. Create a dialog box

9. Place an input widget on the dialog box so the author can enter the list root.

10. Edit ***left.jsp***. Replace the placeholder for *newslist* with a **<cq:include>** of your new component.

Congratulations! You have successfully created a listChildren Component, with a Dialog, that allows content authors to enter content and have it used as a list root. This is a significant step in your development capabilities, as you are now able to create CQ Components that allow you to write content to the repository and use that information to create new rendered content.

Copyright 2010, Day Software AG, Switzerland

Day 3
EXERCISE - Create a Logo Component

Goal

The following instructions explain how to create a Component that allows designers to manage a Web site's logo (binary content), through the use of a Design Dialog. By using a Design Dialog, it will ensure the displayed logo and related link are consistent for all Pages that use the same Template/Design. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

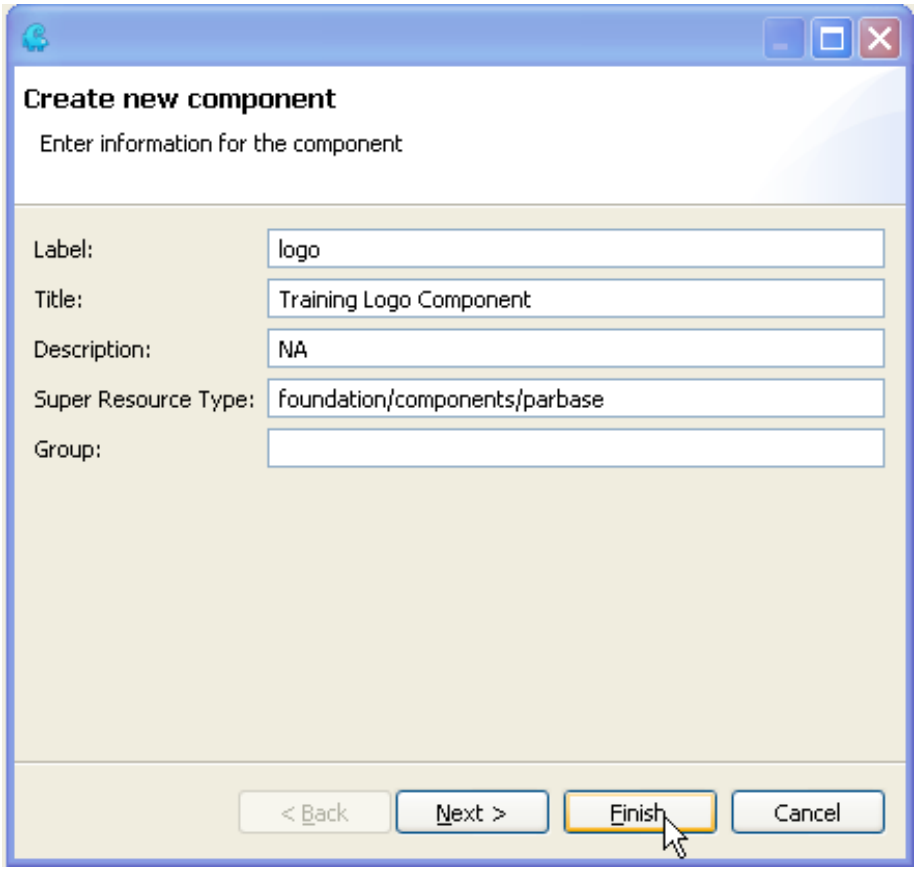
What is special about the Design Dialog?

As discussed earlier, a Design(er) can be used to enforce a consistent look and feel across your Web site, as well as share global content. This global content is editable when using a Design Dialog. So once again, the many Pages that use the same Design(er) will have access to this global content. Fortunately, the process to create a Design Dialog is almost identical to creating a "normal" Dialog – the only difference being the name of the Dialog itself (i.e. dialog vs. design_dialog).

How to create a logo Component with a Design Dialog:

1. Create a new logo "content" Component.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



CRXDE new logo component dialog

NOTE

Important to note is the use of the resource SuperType ***foundation/components/parbase***. Parbase is a key component as it allows components to inherit attributes from other components, similar to subclasses in object oriented languages such as Java, C++, and so on. The parbase here defines tree scripts to render images, titles, and so on, so that all components subclassed from this parbase can use this script.

When using the Dialog Widget smartimage, as you will to complete this exercise, it is recommended to extend this "helper" Component. It contains the Java source img.GET.java, which allows you to more easily manage/transform images in fewer lines of code.

2. Open the file logo.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

logo.jsp

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


Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

```
<%@ page import="com.day.cq.wcm.foundation.Image" %>

<%
    String homePage = "";

    Page parentPage = currentPage.getAbsoluteParent(2);
        if (parentPage == null)
        {
                homePage = currentPage.getPath();
        } else {
              homePage = parentPage.getPath();
        }
%>
<a href="<%= homePage %>.html">
<%
// get the resource of the logo style
log.error("path is: " + currentStyle.getPath());

Resource imageResource = resourceResolver.getResource(currentStyle.getPath
());

if (imageResource != null) {
      // "cast" resource as image
      Image image = new Image(imageResource);

      if (!image.hasContent()) {
          // if image has no content, write out "Home Page"
          %>Home Page<%
      } else {
          // if image has content, set selector, draw out
          image.setSelector(".img"); // use inherited image script
          image.draw(out);
      }
} else {
      %>Home Page<%
}
%>
</a>
```

3. Open the file **left.jsp** in the Training Contentpage Component and replace the “logo” section with a <cq:include> of the logo Component you just created, similar to below – then **Save.**

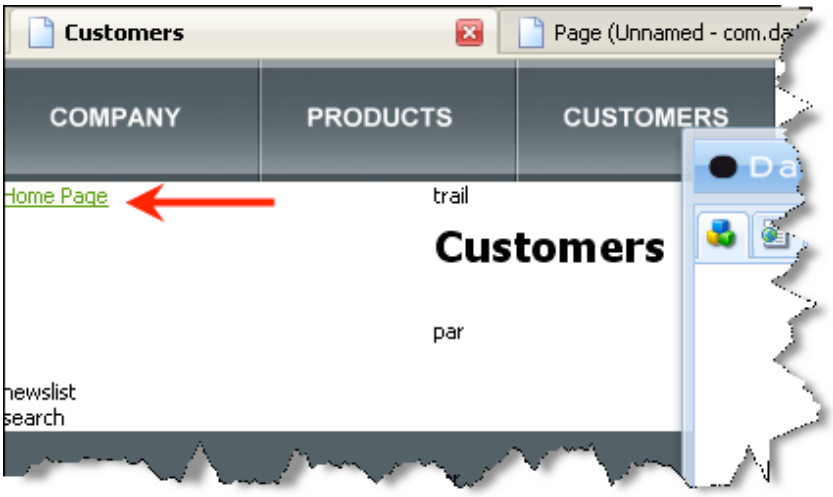
left.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="left">
   <cq:include path="logo" resourceType="training/components/content/logo" />
   <div>newslist</div>
   <div>search</div>
</div>
```


Copyright 2010, Day Software AG, Switzerland

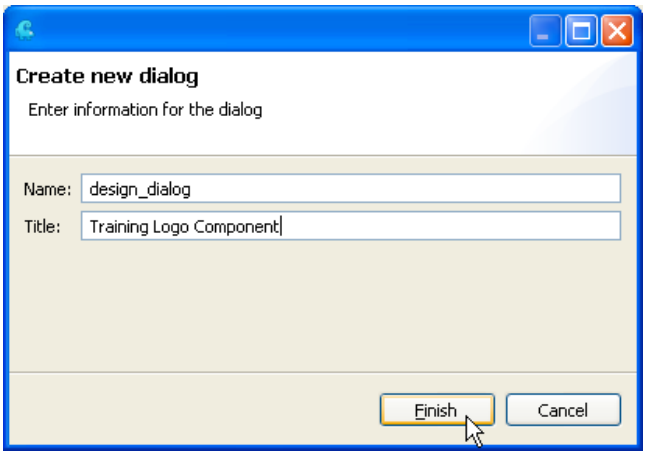
4. Test your script by requesting a Page in CQ5 Siteadmin that implements this Training logo Component.

- If successful, you should see the default logo output, similar to the image below.



Page preview of logo component

5. Create a new Design Dialog for the logo Component.



CRXDE new dialog dialog

6. Create an **items** node (nodeType cq:WidgetCollection) under the *tab1* node of the logo Component's Design Dialog.

Copyright 2010, Day Software AG, Switzerland

7. Create a **margin** node (nodeType cq:Widget) under the newly created *items* node.

8. Assign the following properties to the newly created *margin* node:

- the property that defines where the content is stored
  - Name = name
  - Type = String
  - Value = ./div img.margin
- the property that defines the Widget type
  - Name = xtype
  - Type = String
  - Value = textfield
- the property that defines the label applied to the Widget
  - Name = fieldLabel
  - Type = String
  - Value = Image Margin
- the property that will give the author extra information about the Widget
  - Name = fieldDescription
  - Type = String
  - Value = (e.g., 28px 0 0 48px)

9. Create an **absParent** node (nodeType cq:Widget) under the *items* node.

10. Assign the following properties to the newly created *absParent* node:

- the property that defines where the content is stored
  - Name = name
  - Type = String
  - Value = ./absParent
- the property that defines the Widget type
  - Name = xtype
  - Type = String
  - Value = textfield
- the property that defines the label applied to the Widget
  - Name = fieldLabel
  - Type = String

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- Value = Parent Level (absolute)
- the property that will give the author extra information about the Widget
  - Name = fieldDescription
  - Type = String
  - Value = (e.g., 1 for /content/site)

11. Copy the node /libs/foundation/components/image/dialog/items/image - then paste to the logo's Design Dialog so that it is a peer of *tab1*.

- e.g. /apps/training/components/content/logo/design_dialog/items/items

NOTE

Instead of always reinventing the wheel, it is often more efficient to copy-and-paste existing Dialogs/Widgets that meet your needs. That being said, it is wise to review what you have just copied to better understand the internal workings of CQ.

In addition, you may have noticed this Widget was pasted in the location where tabs are typically located. The smartimage xtype (along with a few others) is unique in that it provides a better content author experience when "casted" as a tab.

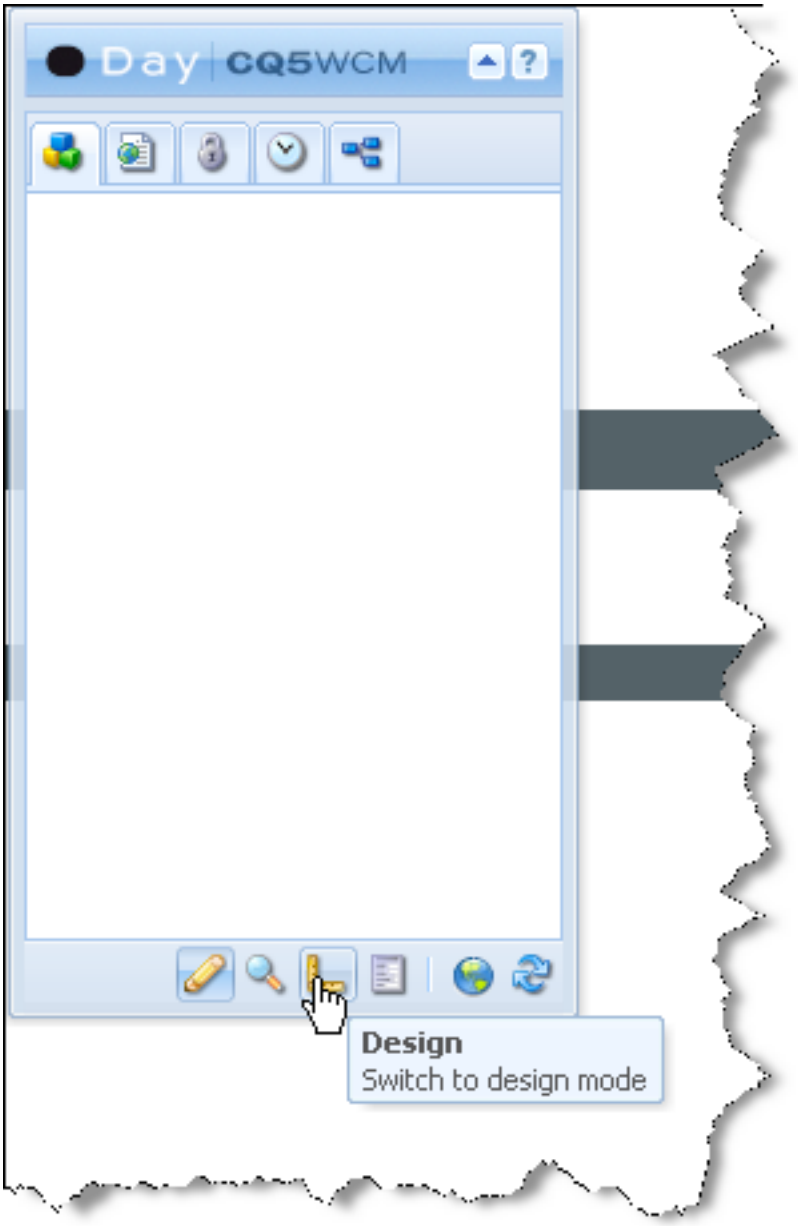
12. Review the properties associated with this ***smartimage*** Widget.

Name	Value	Type
cropParameter	./imageCrop	String
ddGroups	media	String
fileNameParameter	./fileName	String
fileReferenceParameter	./fileReference	String
jcr:primaryType	cq:Widget	Name
mapParameter	./imageMap	String
name	./file	String
renditionSuffix	/_jcr_content/renditions/cq5dam.we...	String
requestSuffix	.img.png	String
rotateParameter	./imageRotate	String
title	Image	String
uploadUrl	/tmp/upload_test/*	String
xtype	smartimage	String

CRXDE properties of image widget

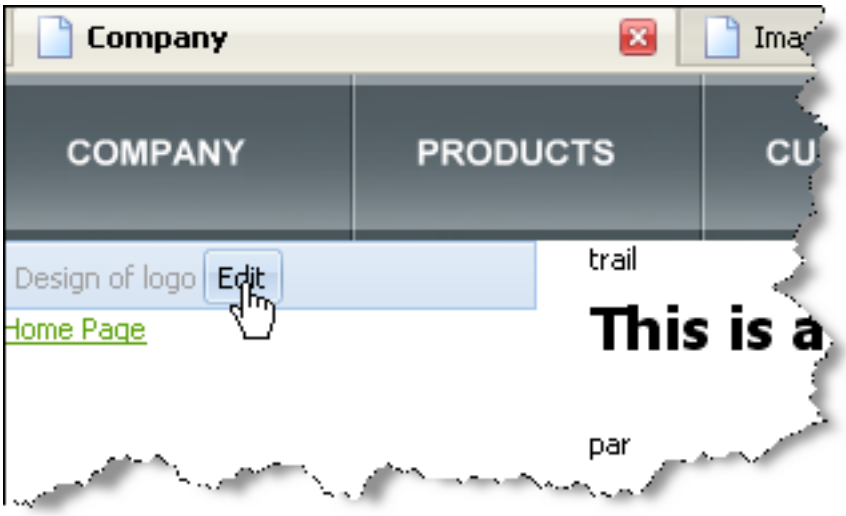
Copyright 2010, Day Software AG, Switzerland

13. Test your script/Design Dialog by requesting a Page in CQ5 Siteadmin that implements this Training logo Component – then select **Design mode** on your Sidekick, and select "edit" for the logo Component to invoke the Design Dialog.



Page design mode selection

Copyright 2010, Day Software AG, Switzerland



Page dialog edit bar

14. Enter the desired custom logo - then select **OK**.

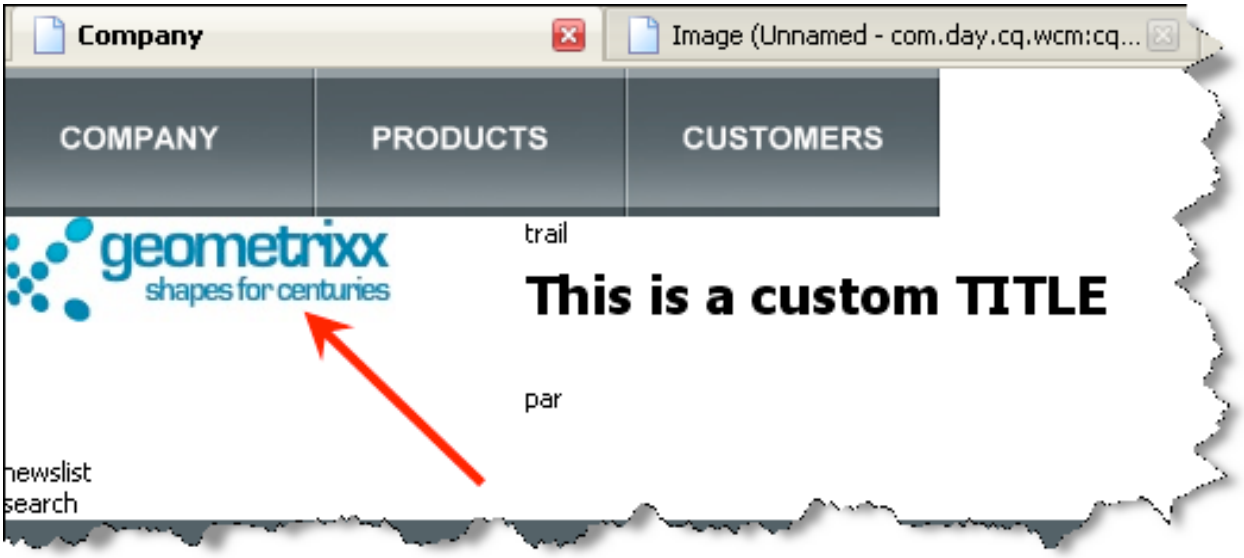
- <usb>/exercises/logo-component/logo.jpg

Design dialog of logo

Copyright 2010, Day Software AG, Switzerland

15. Select **Edit** mode – then review the Page output of the logo Component.

- If successful, you should see the custom Page logo, similar to the image below.



Page preview of logo component

Congratulations! You have successfully created a logo Component, with a Design Dialog, that allows content authors/designers to write global binary content and have it displayed. Once again, this is a significant step in your development capabilities, as you are now able to create CQ Components that allow you to write binary content to the repository. In addition, since this content is global, all pages that implement the same Design/Template will have identical logo content.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Include the Foundation Breadcrumb Component

Goal

The following instructions explain how to include the foundation Component the Breadcrumb, which allows designers to dynamically manage the depth at which a breadcrumb trail begins. This exercise will reinforce your ability to include existing Components into a Template. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

Why include a foundation Component?

Day provides a large amount of Components with its CQ product out-of-the-box. They range anywhere from a simple title Component, to the more complex paragraph system Component, which we will cover later. These Components are located at:

- /libs/foundation/components

It is strongly recommended you perform a complete review of each Component before beginning actual development. Not only are they learning tools, providing excellent examples of what you can do as within CQ as a developer, but you may find an existing foundation Component that provides a completed solution as required by content authors. In addition, you can extend these Components to provide a custom solution, much like you did with the logo Component (which is an extension of the foundation *parbase* Component).

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

How to include the foundation breadcrumb Component:

1. Open the file **center.jsp** in the Training Contentpage Component and replace the "trail" section with a <cq:include> of the foundation breadcrumb Component, similar to below – then **Save**.

center.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<div class="center">
    <cq:include path="trail" resourceType="foundation/components/
breadcrumb" />
    <cq:include path="title" resourceType="training/components/content/
title" />
    <div>par</div>
</div>
```

NOTE

As always, it is a good idea to review what you have just included. Take note of the fact this Component only contains a Design Dialog (i.e. design_dialog), thus can only be modified in Design mode.

2. Test your script by requesting a Page in CQ5 Siteadmin that implements this foundation breadcrumb Component.

- If successful, you should see a breadcrumb trail, similar to the image below.
  - A Cube Page has been added below the Products Page merely for example. Add your own pages below the Products page and open them to test the breadcrumb.
- In Design mode, feel free to modify the depth at which the breadcrumb trail begins.
  - Default is level 2 (i.e. /content/trainingSite/en)

Page preview of breadcrumb

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully included the breadcrumb foundation Component. Again, this is an exercise to get you familiar with not only the process of including a Component into a Template, but also to be more aware of the plethora of Components CQ provides out-of-the-box.

Copyright 2010, Day Software AG, Switzerland

Extra Credit EXERCISE - Modify the Foundation Breadcrumb component

Goal

The following exercise tests your knowledge of creating design elements. The goal is to modify the Foundation Breadcrumb component so that an author with design privileges can set the value of the delimiter between the crumbs. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

Modify the Foundation Breadcrumb component

1. Create the ***foundation/components*** structure in ***/apps*** and copy the Foundation breadcrumb component to the ***.../components*** folder.

2. Use the get of the absParent property as a guide to get the delimiter from the ***currentStyle***.

3. Modify the code to use the delimiter that you got in step 2.

4. Modify the design dialog box to allow the author/designer to enter the delimiter.

5. Test your code.

6. Examine the repository to see the values written into the design when you enter values into the design dialog box and save them.

Congratulations! You have successfully modified the Foundation breadcrumb Component. Again, this is an exercise to use your knowledge to modify a Foundation component.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Extra Credit EXERCISE - Modify your logo component

Goal

The following exercise tests your knowledge of creating design elements. The goal is to modify your local logo component so that the code uses content entered by the author into the existing absParent widget of the design dialog box. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes
- A completed logo Component (e.g. ***training/components/content/logo***)

Modify your logo component

1. Use the breadcrumb component as a guide for getting the ***absParent*** property from the ***currentStyle*** object.

2. Modify the logo component code so that it uses the absParent value to set the level for the link, instead of a hardcoded value.

3. Test your code by entering different values for ***absParent***.

4. Examine the repository to see the values written into the design when you enter values into the design dialog box and save them.

Congratulations! You have successfully modified your logo Component. Again, this is an exercise to use your knowledge of the currentStyle to extract design elements and use them to render content objects.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Extra Credit EXERCISE - Modify your topnav component

Goal

The following exercise tests your knowledge of creating design elements. The goal is to modify your local topnav component so that the code allows an author with design privileges to set the root of the navigation by entering a path. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes
- A completed topnav Component (e.g. ***training/components/content/topnav***)

Modify your topnav component

1. Make whatever modifications that you need to obtain the following result:

   Author/designer can enter a path design element to set the root of the top navigation list of pages.

Congratulations! You have successfully modified your topnav Component. Again, this is an exercise to use your knowledge of the currentStyle to set and extract design elements and use them to render content objects.

Extra Extra Credit

Make a similar modification to your logo component to allow the designer to enter a path for the logo image link.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Include the Foundation Paragraph System Component

Goal

The following instructions explain how to include the foundation Component the Paragraph System, which allows authors to dynamically manage a Page's content. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

What is the foundation Paragraph System Component?

The use of the CQ paragraph system Component is a necessity when wanting to have manageable and scalable Page content, without requiring excessive coding/Template creation. This foundation Component can be located at the path below:

- /libs/foundation/components/parsys

It allows content authors the ability to dynamically add, delete, move, copy, and paste "paragraphs" on a Page, not to mention the ability to use the column control Component to structure your content in columns. In addition, you can decide what "content" Components are allowed to be used by specific instances of the parsys. It is Day's STRONG opinion that the parsys should be used as often as possible, thus allowing for mere simple Component configuration as opposed to creating a large number of Templates that developers then have to maintain.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

How to include the foundation paragraph system Component:

1. Open the file **center.jsp** in the Training Contentpage Component and replace the “par” section with a <cq:include> of the foundation paragraph system Component, similar to below – then **Save.**

center.jsp

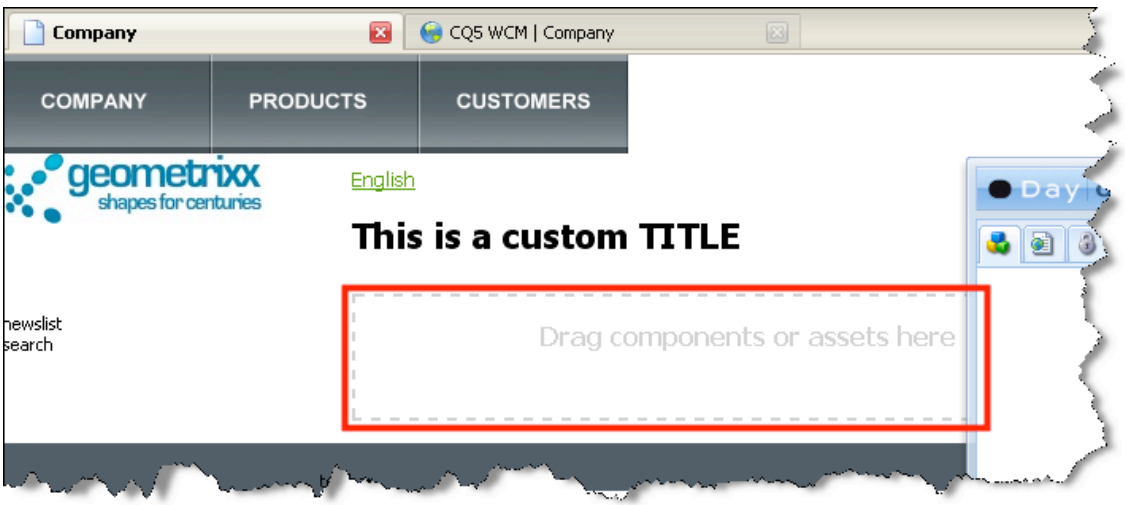
```
<%@include file="/libs/foundation/global.jsp"%>
<div class="center">
    <cq:include path="trail" resourceType="foundation/components/
breadcrumb" />
    <cq:include path="title" resourceType="training/components/content/
title" />
    <cq:include path="par" resourceType="foundation/components/parsys" />
</div>
```

NOTE

As always, it is a good idea to review what you have just included. Take note of the fact this Component only contains a Design Dialog (i.e. design_dialog), thus can only be modified in Design mode.

2. Test your script by requesting a Page in CQ5 Siteadmin that implements this foundation paragraph system Component.

- If successful, you should see the paragraph system enabled, similar to the image below



Page preview of paragraph system

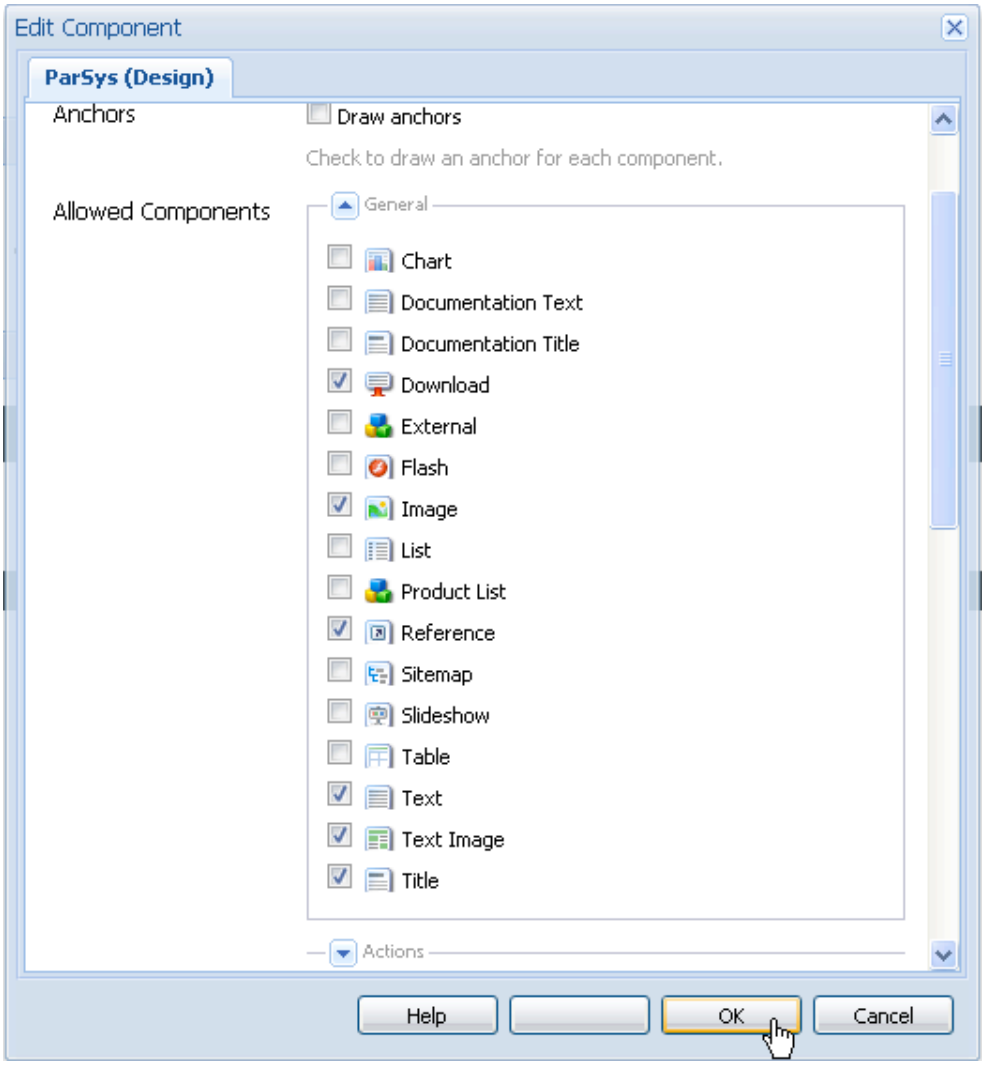
Copyright 2010, Day Software AG, Switzerland

NOTE

You may have noticed earlier, after assigning a new Design to your Web site, that no Components exist (are available) in your Sidekick. This is because you have yet to declare, in Design mode, what Components are allowed to be used by a paragrapgh system. Only then will Components be made available in the Sidekick.

3. Select **Design mode** on your Sidekick – then select "**edit**" for the paragraph system Component to invoke the Design Dialog.

4. Select the Components you would like to allow for this instance of the paragraph system – then select **OK**.

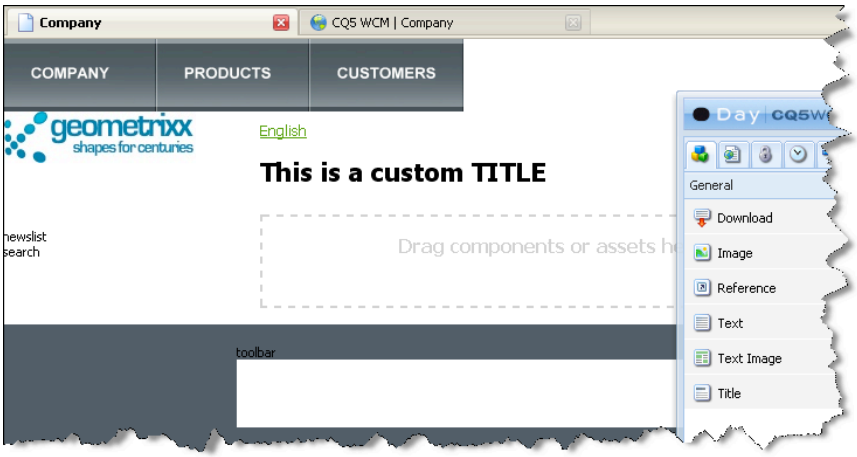


Design dialog of the paragraph system

Copyright 2010, Day Software AG, Switzerland

5. Select **Edit mode** – then review the Page output of the paragraph system Component.

- If successful, you should see the paragraph system and populated Sidekick, similar to the image below.



Page preview of paragraph system

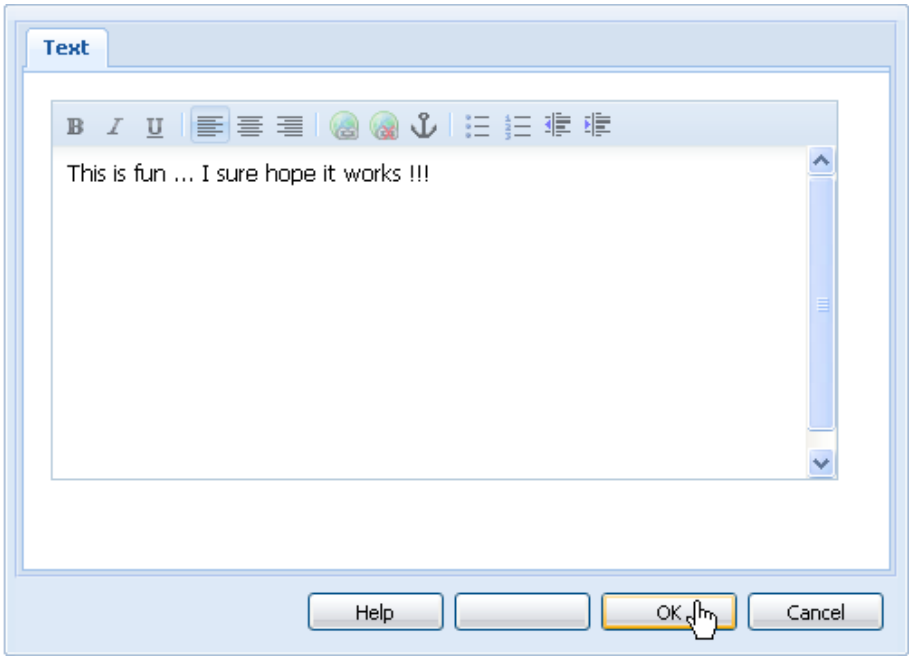
NOTE

Notice how Components now appear in the Sidekick. These Components are now available to use in the paragraph system you recently added to the Template.

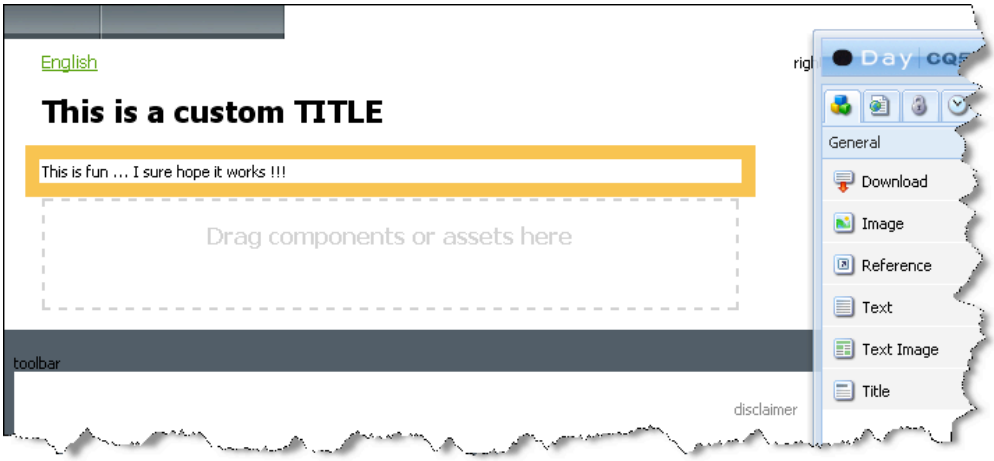
6. Add content (i.e. paragraphs) to the Page by click-and-dragging desired Component(s) into the paragraph system – then write content using the available Dialog.

Page preview of click and drag to paragraph system

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



Dialog of text component



Page preview of written content using the text component

Congratulations! You have successfully included the paragraph system foundation Component. Day cannot stress enough the importance of this Component and how it can greatly affect your productivity if used wisely. Please be sure evaluate its use and optimization for every Template that is developed/ deployed.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Content Finder Drag-and-Drop

Goal

In this exercise we create simple Image component as an example of a component that allows drag-and-drop of assets from the Content Finder onto the paragraph thumbnail placeholder. As a result of this exercise, you will feel more confident building a more complex component that renders binary data and allows drag-and-drop.

To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed "Page" Component (e.g. Training Contentpage) with appropriate includes

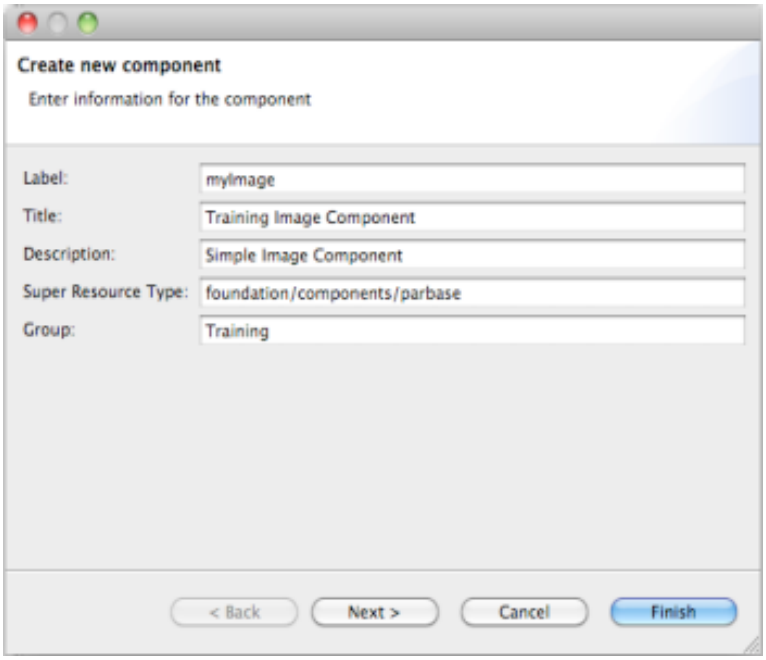
Creating a Simple Image Component

1. Right-click /apps/<application name>/component/content – then select New, Component.

2. Enter the desired Component "Label", "Title", "Description" – then click Finish.

- Label = myImage
- Title = Training Image Component
- Description = Simple Image Component
- Group = Training
- Super Resource Type = foundation/components/parbase
- Allowed Parents = */*parsys

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



3. Enter the following code into **myImage.jsp**:

myImage.jsp

```
<%@ page import="com.day.cq.wcm.foundation.Image" %>
<%@include file="/libs/foundation/global.jsp"%>
<%
    Image image = new Image(resource);

    image.setSelector(".img"); // use image script

    image.draw(out);
%><br>
```

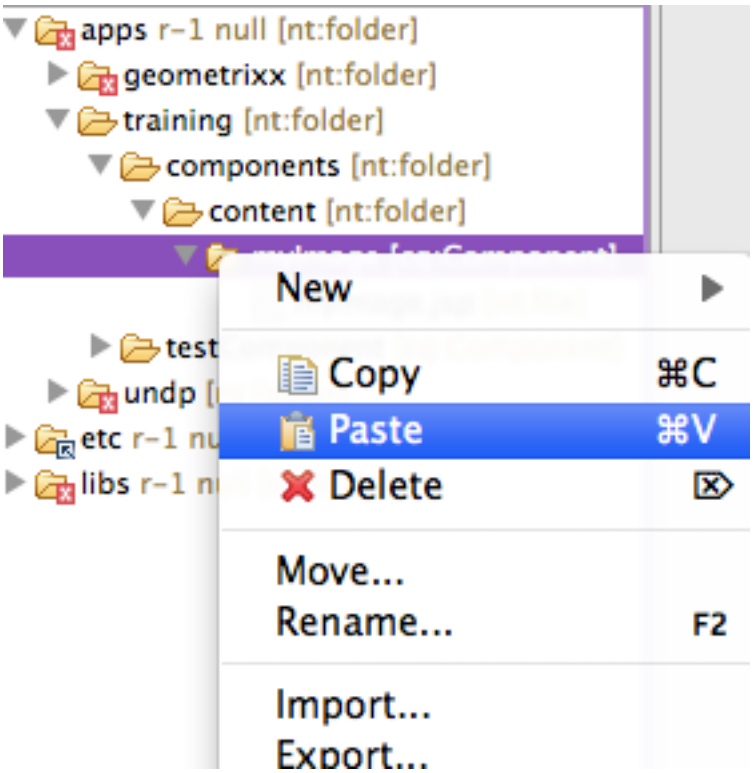
Creating a Dialog Box for our Image component

We need a dialog box, so let's copy the dialog from the **foundation Image** component.

1. Right-click the dialog box from the foundation Image component and select copy.

2. Paste the dialog box in your image component.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



3. Select **Design** mode by clicking on the ruler icon at the bottom of the Sidekick.

4. Click Edit button on **Design of par** edit bar.

5. Select Training Group

6. Return to **Edit mode** by expanding the Sidekick.

Test your Image component.

7. Drag the Training Image component onto a page.

8. Try to drag an image onto the thumbnail. You will note that the drag-and-drop does not work. This is because we have not yet defined the *editConfig* nodes and properties that will support this functionality.

9. Drag-and-drop into the dialog box does work. This is because of the inherent functionality of the smartimage widget. Double-click the thumbnail to open the dialog box.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

10. Drag an image onto the dialog box.

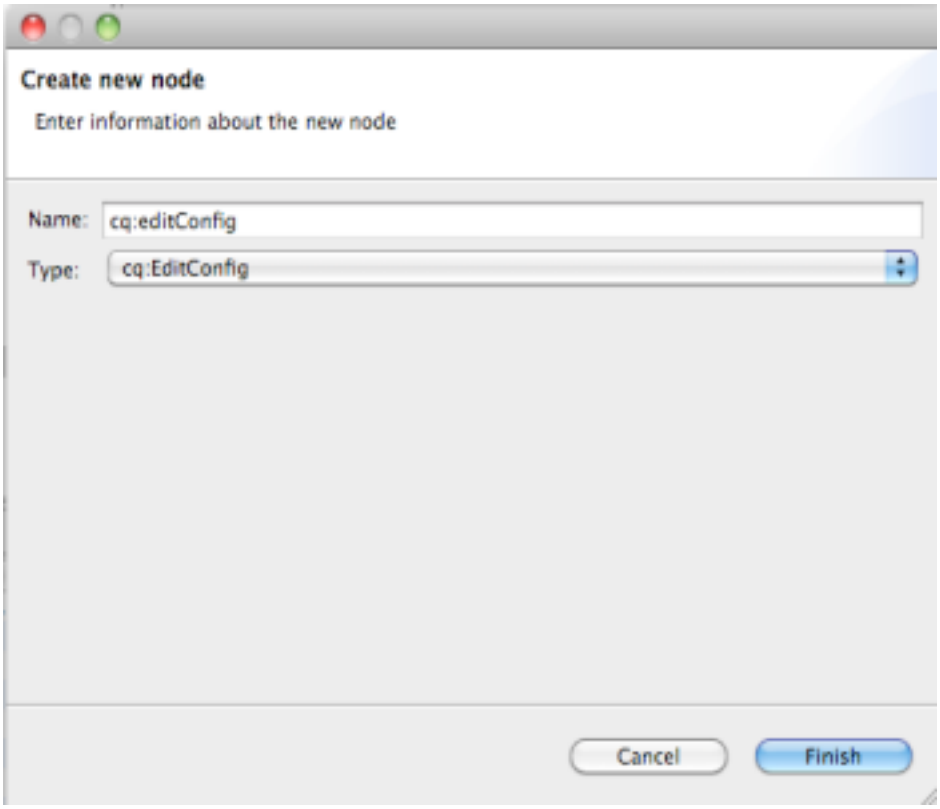
11. Click **OK**.

Defining editConfig

Now that we have our basic component working, let's add an editConfig to allow drag-and-drop from the Content Finder.

1. Right-click on your Image component. Select **New ... Node**.

- Name: cq:editConfig
- Type: cq:EditConfig



2. Right-click the new *cq:editConfig* node and select **New ... Node**.

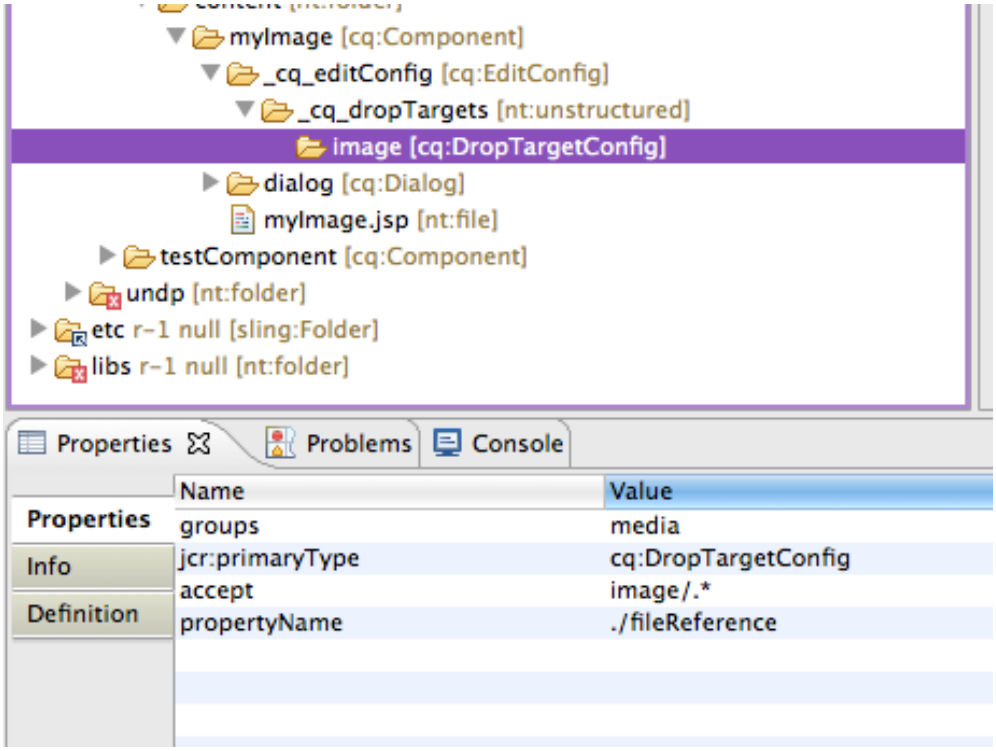
- Name: cq:dropTargets
- Type: nt:unstructured

Copyright 2010, Day Software AG, Switzerland

3. Right-click the new *cq:dropTargets* node and select **New ... Node**

- Name: image
- Type: cq:dropTargetConfig

And then enter the *groups*, *accept*, and *propertyName* properties as shown in the diagram below:



4. Test the new functionality by dragging a new Training Image component onto the page and then dragging an image from the Content Finder onto the new paragraph.

Congratulations! You have successfully created a Simple Image component that allows drag-and-drop from the Content Finder. As an exercise, examine a few of the other components in /libs/foundation/components that support drag-and-drop from the Content Finder. For example, the Download, Slideshow, and List components.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create a Complex Component

Goal

The following instructions explain how to create a "complex" Component that manages both textual and binary (i.e. image) content, and has both a Dialog and Design Dialog. This includes allowing the Component to be used by the parsys Component, creating a Dialog with multiple tabs, enabling the functionality offered by the Content Finder (i.e. drag-and-drop), and other configurations. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes (parsys)

When will I need to create a "complex" Component?

Since the creation of "complex" Components is a common occurence in CQ, it would benefit you to observe a mock requirements analysis. This exercise provides just that. First, you will observe the needs of a user. Secondly, you will observe how a Day Solution Engineer may translate those needs.

USER'S NEEDS: A Component that can be dropped into a paragraph system and displays an image, rich text, and the path of a Page in the system.

The image:

1. Must be editable by a content author.
2. Can be dragged-and-dropped from the Content Finder

The rich text:

1. Must be editable by a content author
2. Must allow for tables to be created.
3. Must have a default value of "This is some text."

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

The path of a Page:

1. Must be the same for every instance of this Component, yet editable by a "super" author.
2. Must live under the Web site structure "/content/trainingSite".

SOLUTION ENGINEER'S TRANSLATION: A paragraph system Component that allows for the writing and displaying of three properties (2 paragraph properties, 1 design/style property), and has both a Dialog and Design Dialog.

The image:

1. Is a Dialog Widget, most likely an xtype of smartimage.
2. Must be configured in the Component's cq:editConfig to allow for dragging-and-dropping of images from the Content Finder.

The rich text:

1. Is a Dialog Widget, most likely an xtype of richtext.
2. Widget should enable all the features of the rich text editing plugin table.
3. Widget should populate property defaultValue with "This is some text."

The path of a Page:

1. Is a Design Dialog Widget, most likely an xtype of pathcompletion.
2. Widget should have property regex with a regular expression validating the user's input (e.g. "/^\/content\/training\/(.)*$/").
3. Widget should have property regexText with an error message if regular expression fails.

Though the order of sequence can differ, a Day Solution Engineer would most likely:

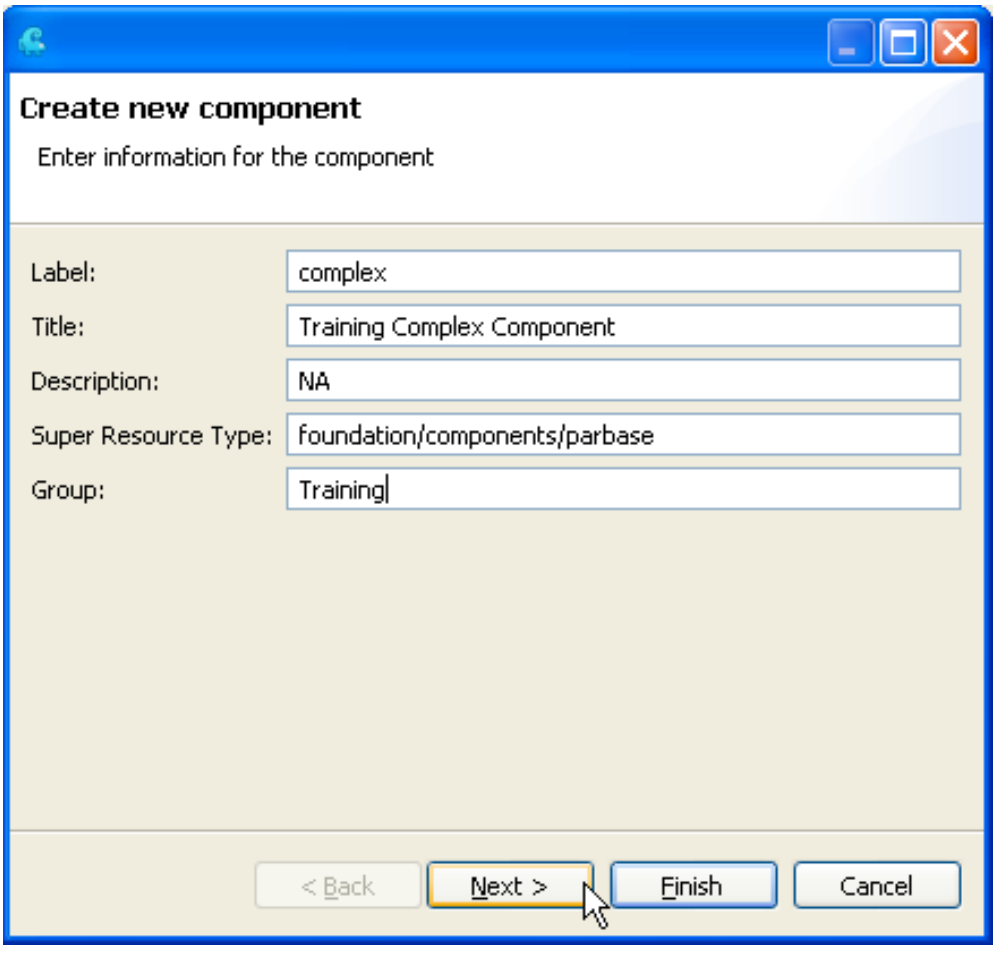
1. Create a Component, making sure to allow that any paragraph system Component can be this Component's "parent."
2. Edit the default JSP so it displays the content/properties in an appropriate manner, even without any written content.
3. Create a Dialog for the Component.
4. Create a Design Dialog for the Component.
5. Add this Component to the list of allowed Components for a paragraph system Component.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

6. Test this Component by adding it to a Page to observe its output without any written content.
7. Add a rich text Widget to the Dialog.
8. Add an image Widget to the Dialog.
9. Add a path completion Widget to the Design Dialog.
10. Perform necessary Component configurations.
11. Test this Component by writing content using the newly created Dialog and Design Dialog.

How to create a complex Component using CRXDE:

1. Create a new complex "content" Component.

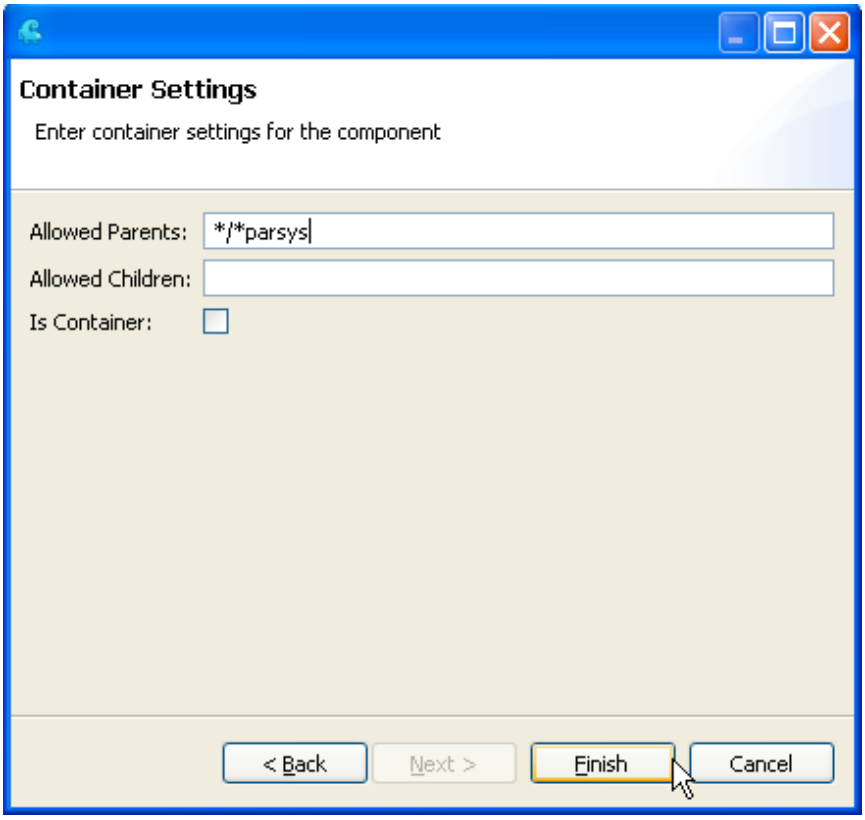


CRXDE new complex component dialog - 1

Copyright 2010, Day Software AG, Switzerland

NOTE

Notice how the pointer on the image above is selecting ***Next***. This is necessary so that you can declare this Component has an allowed parent with the next dialog.



CRXDE new complex component dialog - 2

2. Open the file **complex.jsp**, and enter some HTML and JSP code, similar to below – then **Save**.

complex.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="com.day.cq.wcm.foundation.Image" %>
<%
// getting the image from the Dialog/resouce
// notice the use of the second parameter "image" -- this signifies that the
// image will live on a resource (called image) below the requested resource
Image image = new Image(resource, "image");
// setting the selector so that the "parbase" can work its magic
image.setSelector(".img");
// getting the rich text from the Dialog
String text = properties.get("text", "TEXT NA");
// getting the path from the Design Dialog
String path = currentStyle.get("path", "PATH NA");
```


Copyright 2010, Day Software AG, Switzerland

```
%>
<h2><%= path %></h2>
<%= text %><br />
<%
image.draw(out);
%>
```

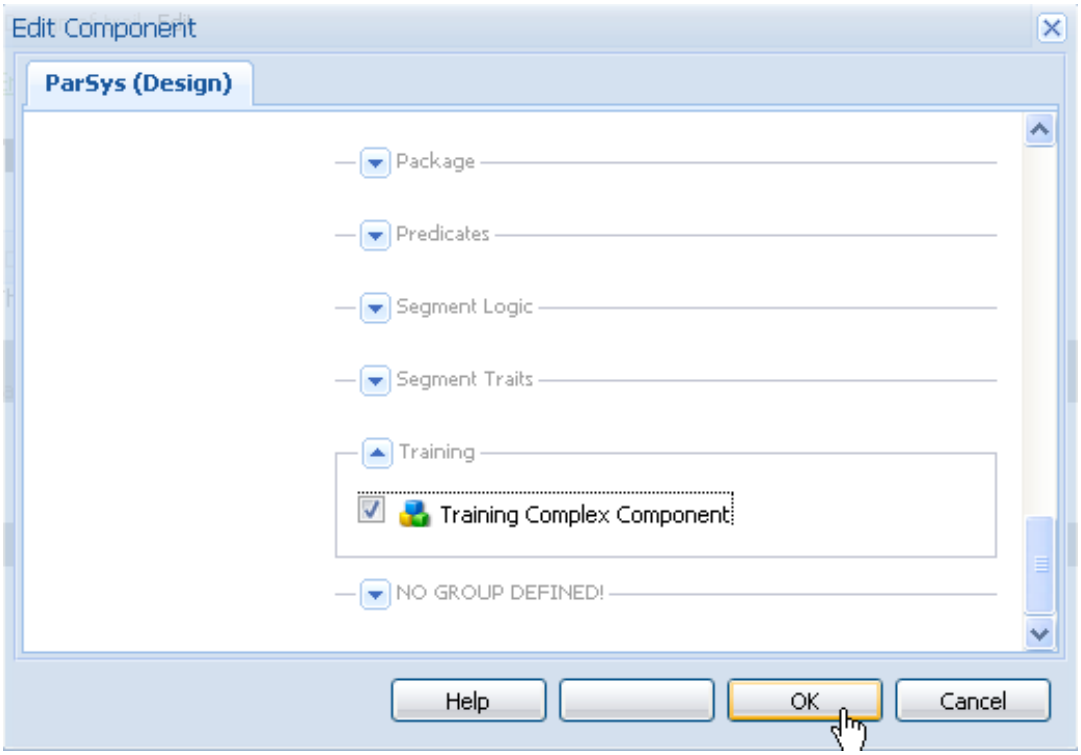
NOTE

For this exercise, do not concern yourself greatly with how the content is displayed, as this can easily be altered via code changes and/or CSS.

3. Create a Dialog and Design Dialog for the complex Component.

- You will worry about Widgets and configurations later – the focus now is to see your new Component in action.

4. Add your complex Component to the paragraph system Component in Design mode.



Design dialog of the paragraph system

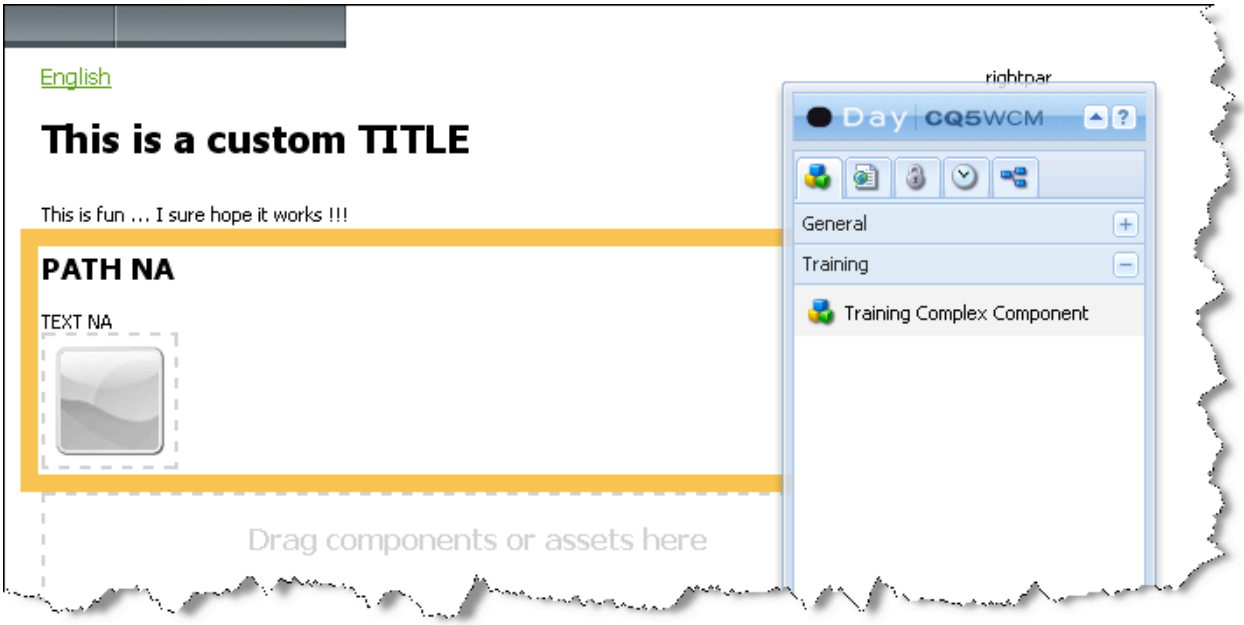
Copyright 2010, Day Software AG, Switzerland

NOTE

Notice how your complex Component is listed under the Training group. This is because you declared such during the creation of your Component.

5. Test your script by adding an instance of this Component (i.e. paragraph) to the paragraph system Component of a Training Page.

- If successful, you should see the default complex Component output, similar to the image below.



Page preview of complex component - no content

6. Create an **items** node (nodeType cq:WidgetCollection) under the *tab1* node of your Component's Dialog.

7. Create a **text** node (nodeType cq:Widget) under the newly created *items* node.

8. Assign the following properties to the newly created *text* node:

- the property that will define where content is stored
  - Name = name
  - Type = String
  - Value = ./text

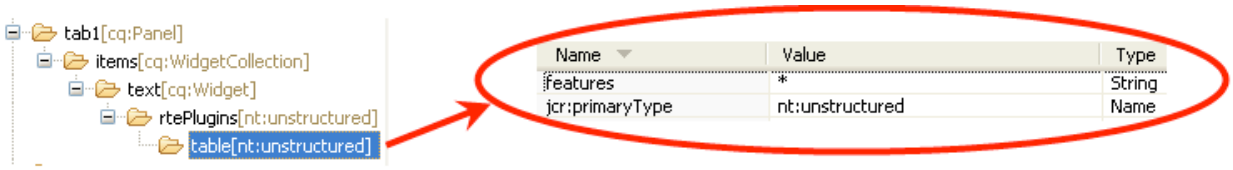
Copyright 2010, Day Software AG, Switzerland

- the property that will define the Widget type
  - Name = xtype
  - Type = String
  - Value = richtext
- the property that will define to hide the label of the Widget
  - Name = hideLabel
  - Type = Boolean
  - Value = true
- the property that will define the the default value of the Widget
  - Name = defaultValue
  - Type = String
  - Value = This is some text.

9. Create an **rtePlugins** node (nodeType nt:unstructured) under the newly created *text* node.

10. Create a **table** node (nodeType nt:unstructured) under the newly created *rtePlugins* node.

11. Assign the features property (Type = String, Value = *) to the newly created *table* node.



CRXDE retPlugin table plugin structure and property

12. Copy the nodes *tab2* and *tab3* under the node */libs/foundation/components/textimage/dialog/items* – then paste to the complex Component's Dialog so that they are a peer of *tab1* (e.g. /apps/training/components/content/complex/dialog/items/items).

- tab2 = the smartimage tab
- tab3 = advanced image properties

Copyright 2010, Day Software AG, Switzerland

NOTE

Once again, it is often more efficient to copy-and-paste existing Dialogs/Widgets that meet your needs. That being said, it is wise to review what you have just copied to better understand the internal workings of CQ.

If you examine closely enough, you will see the Widget is actually storing image related content at a level deeper than current resource (e.g. ./image/file, ./image/fileReference, etc.). This ties nicely with your previously written code (Image image = new Image(resource, "image");).

13. Now we build out the Design Dialog. Create an **items** node (nodeType cq:WidgetCollection) under the *tab1* node of your Component's Design Dialog.

14. Create a **path** node (nodeType cq:Widget) under the newly created *items* node.

15. Assign the following properties to the newly created *path* node:

- the property that will define where content is stored
  - Name = name
  - Type = String
  - Value = ./path
- the property that will define the Widget type
  - Name = xtype
  - Type = String
  - Value = pathcompletion
- the property that will define the label applied to the Widget
  - Name = fieldLabel
  - Type = String
  - Value = Path of a page
- the property that will define the root path to display
  - Name = rootPath
  - Type = String
  - Value = /content/trainingSite
- the property that will define the regular expression used to evaluate user input
  - Name = regex
  - Type = String
  - Value = /^\/content\/trainingSite\/(.)*$/

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- the property that will define the error message if a user's input fails the regular expression
  - Name = regexText
  - Type = String
  - Value = Please insert a Page that "lives" under /content/trainingSite

16. Copy the node /libs/foundation/components/textimage/cq:editConfig - then paste to the root node of your complex Component (e.g. /apps/training/components/content/complex).

- enables drag-and-drop capabilities from the Content Finder

In order to be able to drag-and-drop assets from the Content Finder to a Component on a Page, there must be a drop targets configuration node called cq:dropTargets (of type nt:unstructured) below the edit configuration node (cq:editConfig) of a Component.

17. Using CRXDE, navigate to

<path-to-component>/cq:editConfig/cq:dropTargets/image

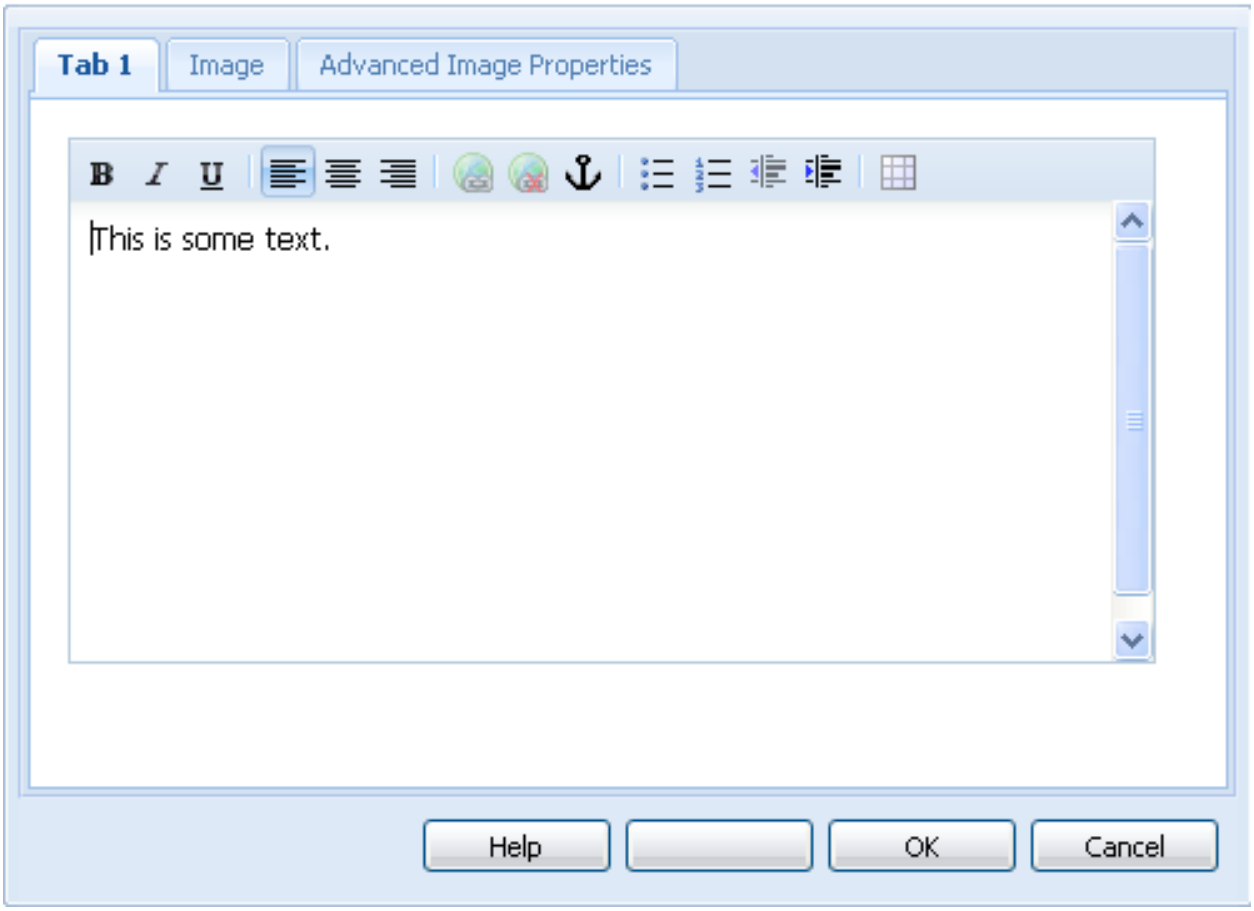
Validate that the image node has the following properties:

- accept (Type = String) - the media types to be accepted (e.g. image/.*, etc.)
- groups (Type = String) - the groups in the Content Finder assets can be accepted from (e.g. media)
- propertyName (Type = String) - the property the reference should be stored (e.g. ./image/fileReference)

18. Test your script/Dialog by requesting a Page in CQ5 Siteadmin that implements this Training complex Component - then interact with the Dialog and Design Dialog.

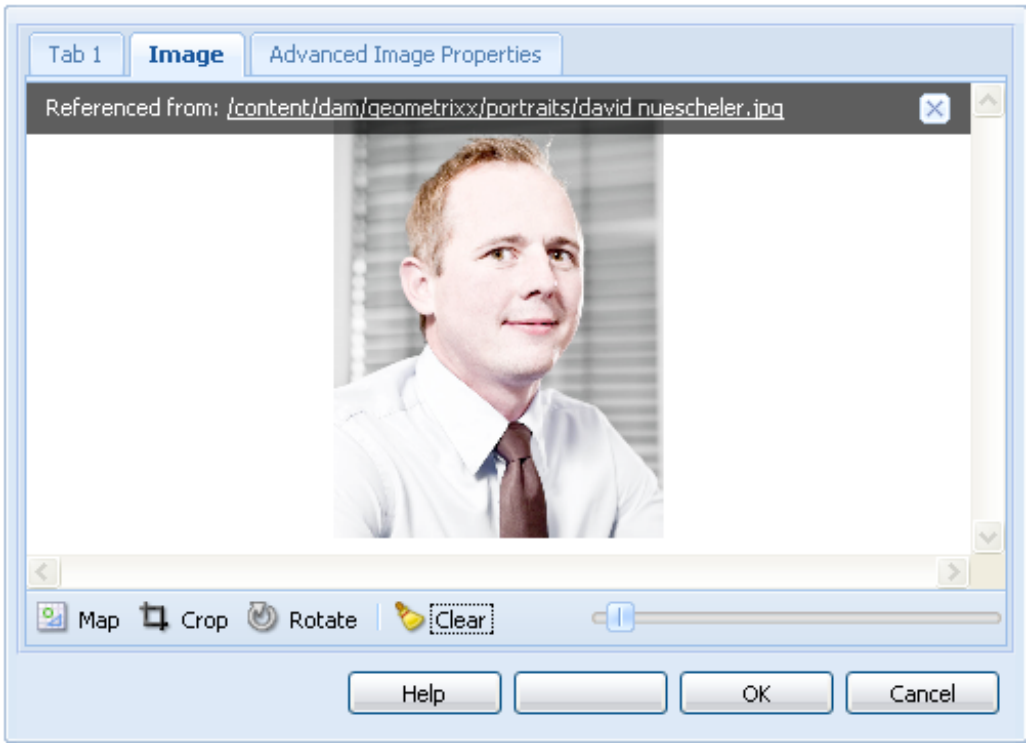
- If successful, you should see Page output, a Dialog and Design Dialog similar to the images below.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

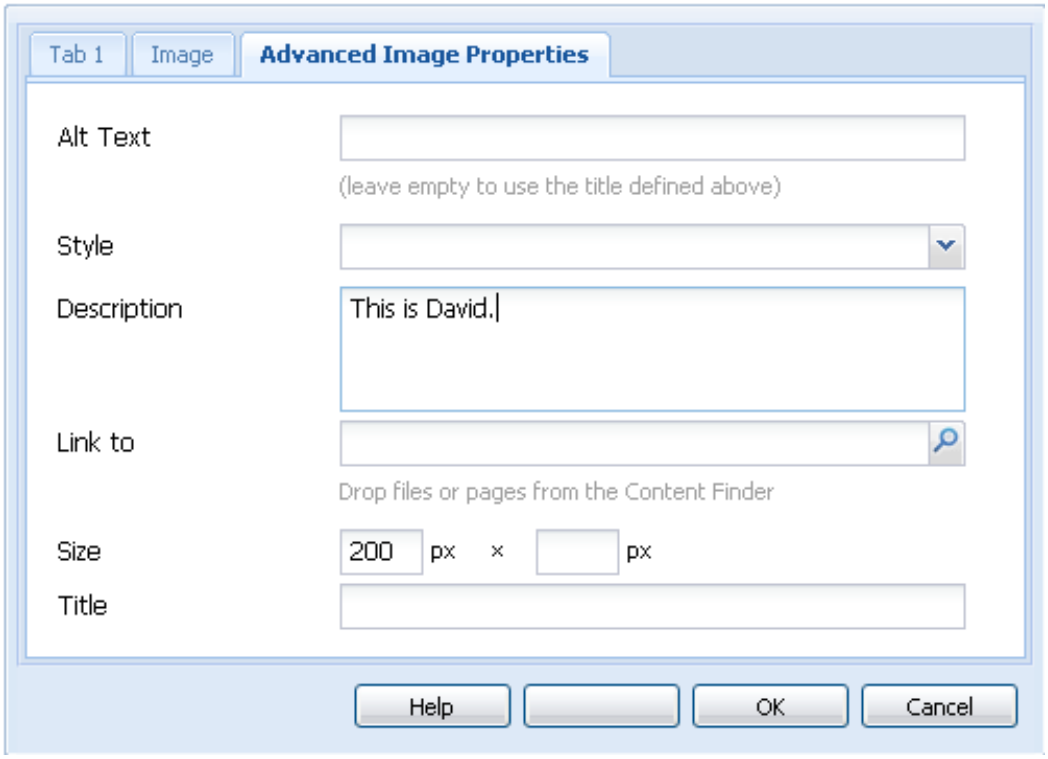


Dialog of complex component - 1

Copyright 2010, Day Software AG, Switzerland

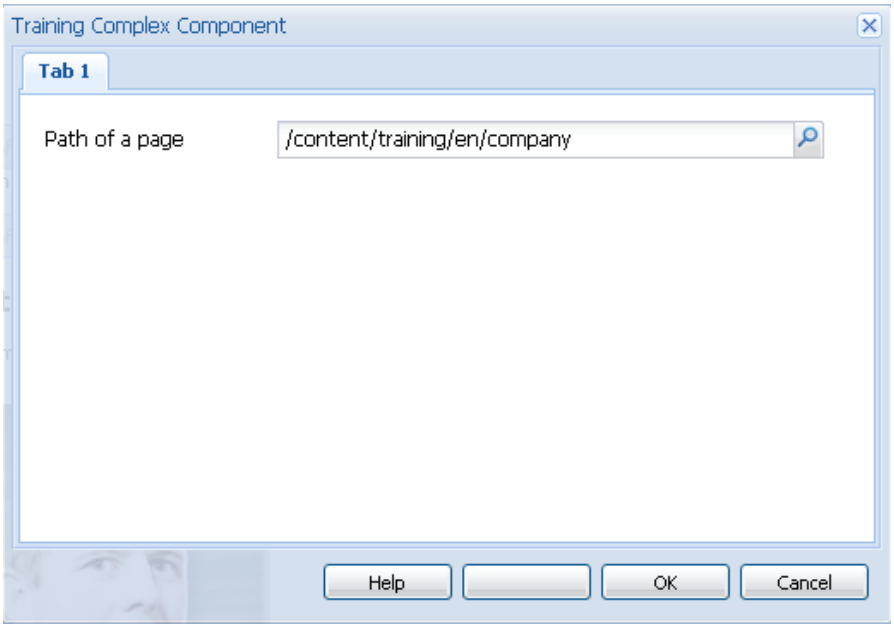


Dialog of complex component - 2

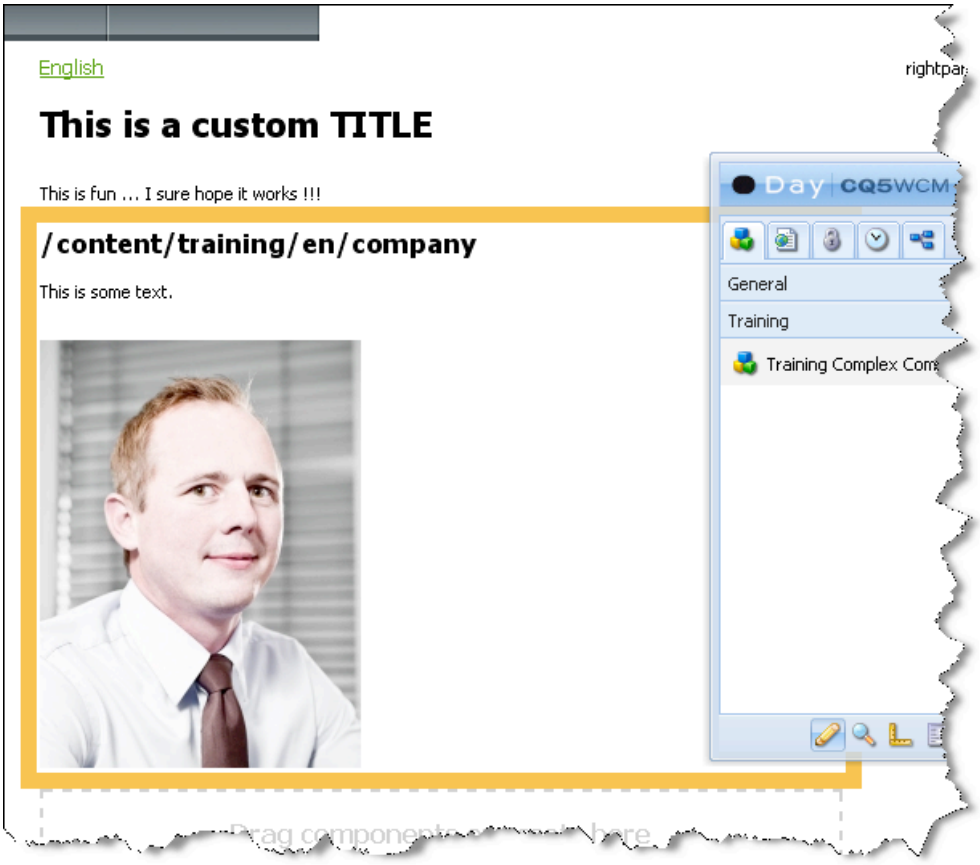


Dialog of complex component - 3

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



Design dialog of complex component - 3



Page preview of complex component - with content

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully created a complex Component that contains a Dialog, Design Dialog, default values, custom configuration, and validation of user input, in addition to enabling drag-and-drop functionality from the Content Finder. Give yourself a pat on the back - this was quite a challenge.

That being said, it would benefit you to be aware of additional cq:editConfig possibilities/variations. For example:

- cq:inplaceEditing - this is where you allow content authors to have in-context-editing for textual content
  - review /libs/foundation/components/text
- cq:listeners - this is where you can define actions (e.g. REFRESH_PAGE, REFRESH_SELF, custom JS function, etc.) based on Component events (e.g. afterinsert, afteredit, aftercopy, etc.)
  - review /libs/foundation/components/slideshow

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Include Multiple Foundation Components

Goal

The following instructions explain how to include multiple foundation Components. This exercise is used to apply finishing touches to the look and feel of the Training "Page" Component. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes

Finishing the layout:

Including the Foundation Toolbar component

1. Open the file ***body.jsp*** in the Training Contentpage Component and replace the “toolbar” section with a <cq:include> of the foundation toolbar Component – then **Save.**

   - path = “toolbar”
   - resourceType = “foundation/components/toolbar”

2. In the Websites tab (Site Admin Console), select the English page under Training Site. Create the Toolbar page under English. Using the Page Properties... in Sidekick set Hide in Navigation (in Sidekick) to prevent the Toolbar page and all of its children from appearing in the Navigation component.

3. Under Toolbar, create the following pages:
   - Contacts
   - Feedback
   - Login
   - Search

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

When you load any of the pages in your website, the toolbar will look as follows:

Including the Foundation Timing component

The Foundation Timing component will record data on component script execution time. We will use the results of this component in the *Performance Optimization* section of the course.

1. Insert a <cq:include> of the foundation timing Component directly above the </body> element in the file body.jsp – then **Save.**

   - path = "timing"
   - resourceType = "foundation/components/timing"

Including the Foundation Inherited Paragraph System component

The inherited paragraph system (***iparsys***) is a paragraph system that also allows you to inherit the created paragraphs from the parent page. You add paragraphs to iparsys at for example, */content/trainingSite/en.html* and as result, all the subpages of *English* that also have an ***iparsys*** with the same name inherit the created paragraphs from the *English* parent. On each level, you can add more paragraphs, which are then inherited by the children pages. You can also cancel paragraph inheritance at a level at any time.

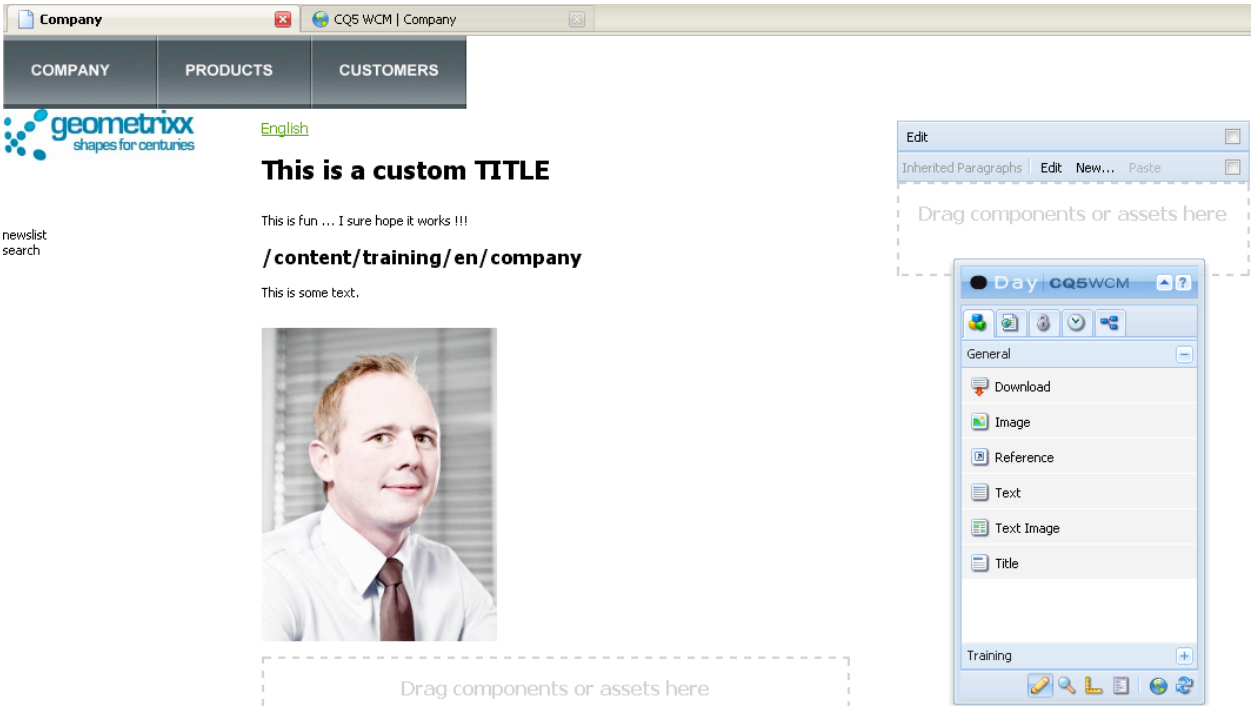
1. Open the file right.jsp in the Training Contentpage Component and replace the "rightpar" section with a <cq:include> of the foundation inherited paragraph system (i.e. iparsys) Component – then **Save.**

   - path = "rightpar"
   - resourceType = "foundation/components/iparsys"

Copyright 2010, Day Software AG, Switzerland

2. Test your script by requesting a Page in CQ5 Siteadmin that implements these multiple foundation Components.

- If successful, you should see a Page view similar to the image below.



Page preview of completed page component

Congratulations! You have successfully included multiple foundation Components. Again, this exercise is used to apply finishing touches to the look and feel of the Training "Page" Component.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create a Search Component

Goal

The following instructions explain how to create a Component that will allow visitors to search the content of the Web site/repository. This search Component can be placed in the parsys of any Page, and has the ability to search the content of the Web site based on a query string provided in the request. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed “Page” Component (e.g. Training Contentpage) with appropriate includes (parsys)

How can I search Web site content?

Searching for content in CQ is very similar to "traditional" searches you have created in the past.

1. An HTML form is needed to collect the user's input (search string).
2. Once the form has been submitted, you need to capture the search string.
3. You need to prepare a query statement based on the search string.
4. A query object needs to be created that will connect to the content repository, and implement the query statement.
5. You need to collect and parse the query results, if any.
6. Output related to the query results should be displayed appropriately.

When querying a Java Content Repository (JCR), some basic functionality (API calls) need to be implemented:

- javax.jcr.Session – JCR session, can be reached for example by Node.getSession()
- javax.jcr.Workspace – JCR workspace, can be reached for example by Session.getWorkspace()

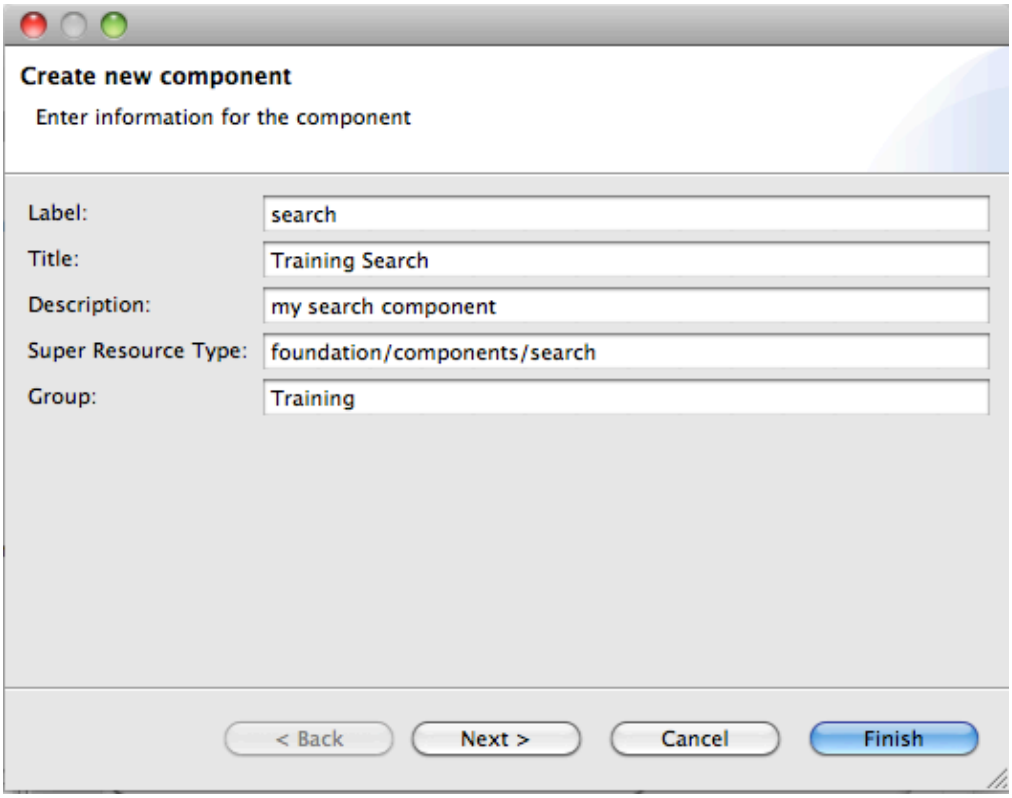
Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- javax.jcr.query.QueryManager – QueryManager is used to create a Query object and can be reached via Workspace.getQueryManager()
  - createQuery(String statement, String language) – creates the Query object for the provided statement in the provided language (e.g. SQL)
- javax.jcr.Query
  - execute() – executes this Query and returns a QueryResult object
- javax.jcr.query.QueryResult
  - getRows() – returns an iterator over the Rows of the query result table

For more detailed information, please review the Javadoc of the JCR and CRX provided on the USB.

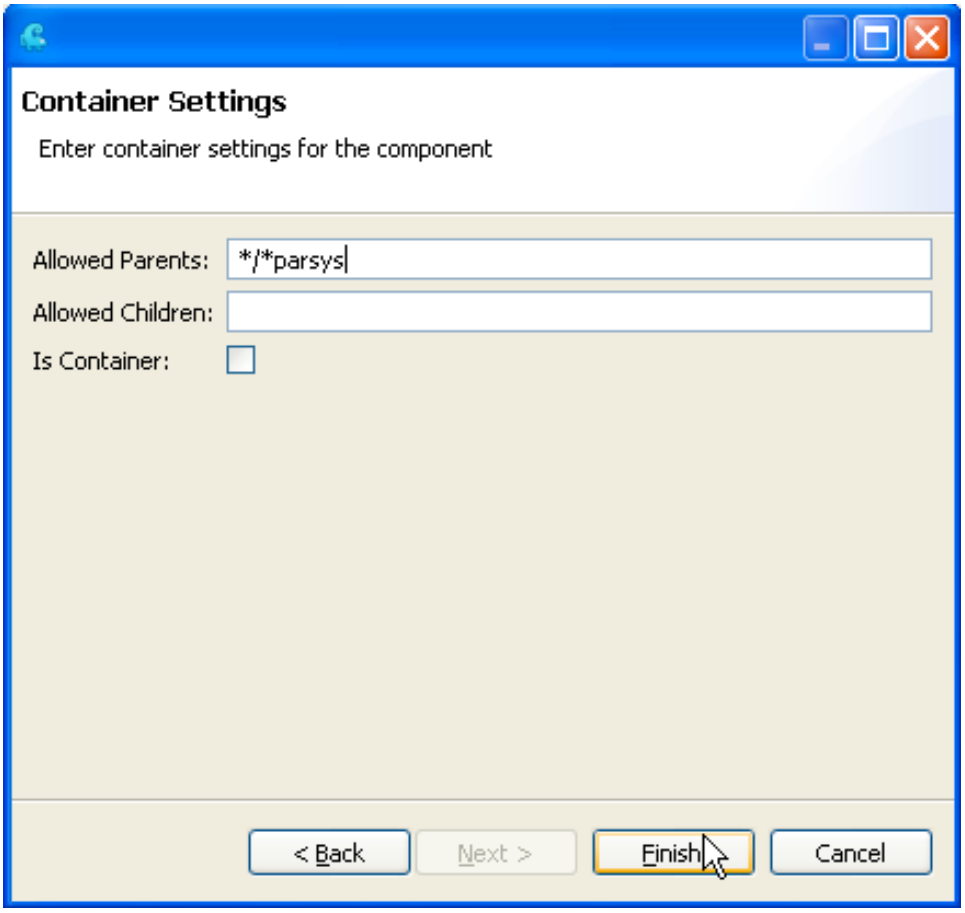
How to create a search Component:

1. Create a new search "content" Component.



CRDXE new search component dialog - 1

Copyright 2010, Day Software AG, Switzerland



CRDXE new search component dialog - 2

2. Open the file search.jsp, and enter some HTML and JSP code, similar to below – then **Save**.

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="javax.jcr.query.QueryResult,
                 javax.jcr.query.Query,
                 javax.jcr.Node,
                 javax.jcr.NodeIterator,
                 org.apache.commons.lang.StringEscapeUtils" %>
<%
String queryString = (slingRequest.getParameter("q") != null) ?
StringEscapeUtils.escapeXml(slingRequest.getParameter("q")) : "";
%>
<center>
      <form action="<%= currentPage.getPath() %>.html">
         <input name="q" value="<%= queryString %>" />
         <input value="Search" type="submit" />
      </form>
</center>
<br />
```


Copyright 2010, Day Software AG, Switzerland

```
<%
if (slingRequest.getParameter("q") != null) {
      String stmt = "select * from cq:Page where contains(*, '" +
slingRequest.getParameter("q") + "') order by jcr:score desc";
      Query query = currentNode.getSession().getWorkspace().getQueryManager
().createQuery(stmt, Query.SQL);
      QueryResult results = query.execute();
      if (results.getNodes() != null && results.getNodes().hasNext()) {
            NodeIterator it = results.getNodes();
            while (it.hasNext()) {
                  Node node = it.nextNode();
                  String npath = node.getPath();
                  Page contentPage = pageManager.getContainingPage
(resourceResolver.getResource(npath));
                  String title = contentPage.getTitle();
                  String path = contentPage.getPath() + ".html";
                  %>
                  <div class="searchresult"><a href="<%= path %>"><%= title
%></a></div>
                  <%
            }
      } else {
            %>
            <div class="searchresult">No results found ... Please try
again ...</div>
            <%
      }
}
%>
```

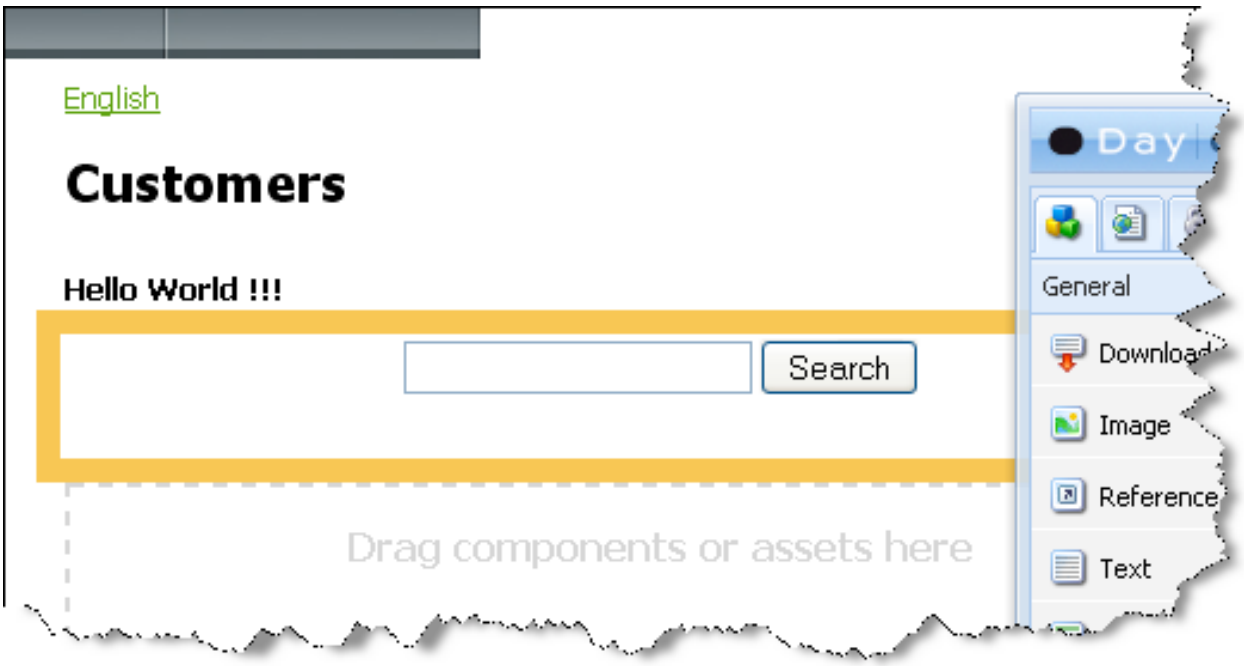
3. Add your search Component to the paragraph system Component in **Design mode**.

- Notice how the Component will be found in the "Training" group – this was defined during the creation of the Component

4. Test your script by adding an instance of this Component (i.e. paragraph) to the paragraph system Component of a Training Page.

- If successful, you should see the default complex Component output, similar to the image below.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

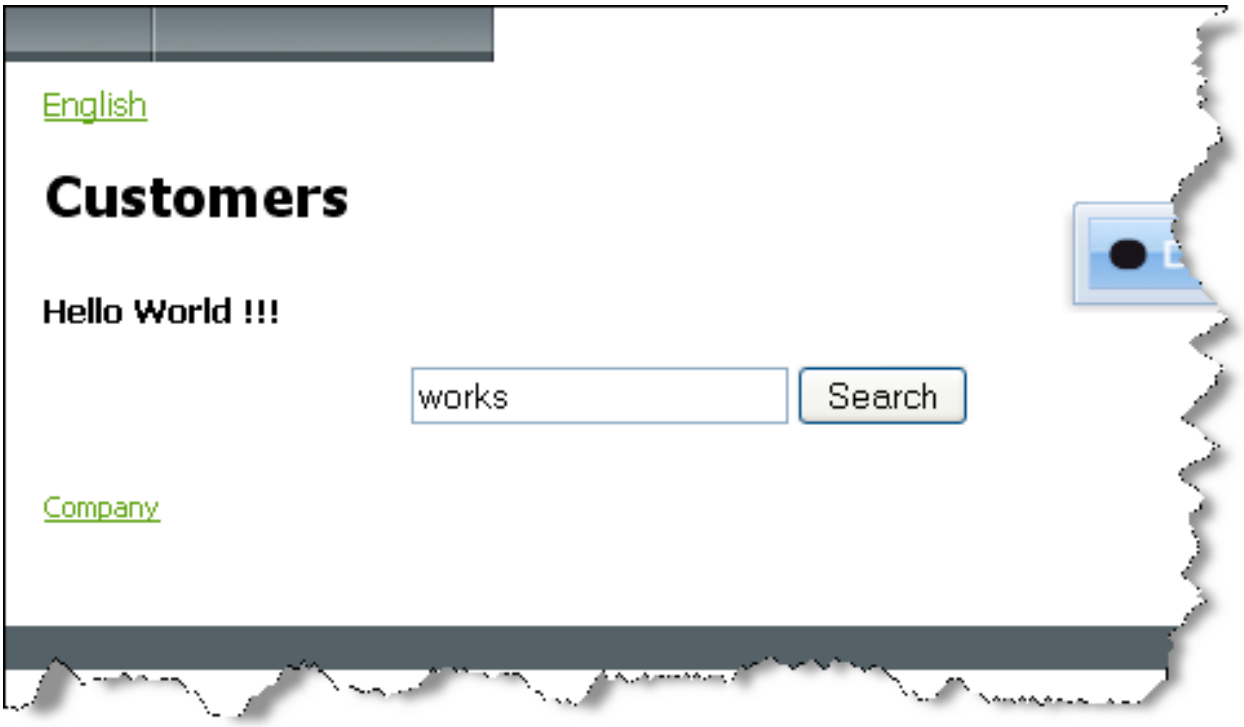


Page preview of search component

5. Search for a word you know exists on a separate Page in your Training Web site structure.

- If successful, you should see search results output, similar to the image below
- You may need to perform this search in Page preview mode, to ensure a "clean" request

Copyright 2010, Day Software AG, Switzerland



Page preview of search results

Congratulations! You have successfully created a search Component that queries the content in your Training Web site structure. You can further enhance this Component by adding Widgets to the Dialog to output default messages, written by a content author, if the search was successful, or unsuccessful.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Apply i18n to a Component

Goal

The following instructions explain how to apply CQ's internationalization (i18n) capabilities in a Dialog. This will allow you to provide dynamic Dialog "messages" based on the authors' language preference. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed title Component

What is i18n and when should I use it?

As stated earlier, i18n allows you to provide dynamic messaging based on the authors' language preference. Internationalization is stored in the repository under nodes (nodeType sling:Folder) named i18n. The children nodes (nodeType sling:Folder + mixin mix:langauge) of the i18n node represent languages and are named after the ISO code of the languages that should be supported (e.g. en, de, fr, etc.). Below these language nodes are the message nodes (nodeType sling:MessageEntry), which will contain a simple key-message pair.

The location of the i18n node within the repository decides its scope. If located in aproject directory (e.g. /apps/training), it should contain only messages related to the current project. However, if located in a Component hierarchy, it should contain only Component specific translations. Globally used messages should be place in /apps/i18n.

How to apply i18n to the title Component:

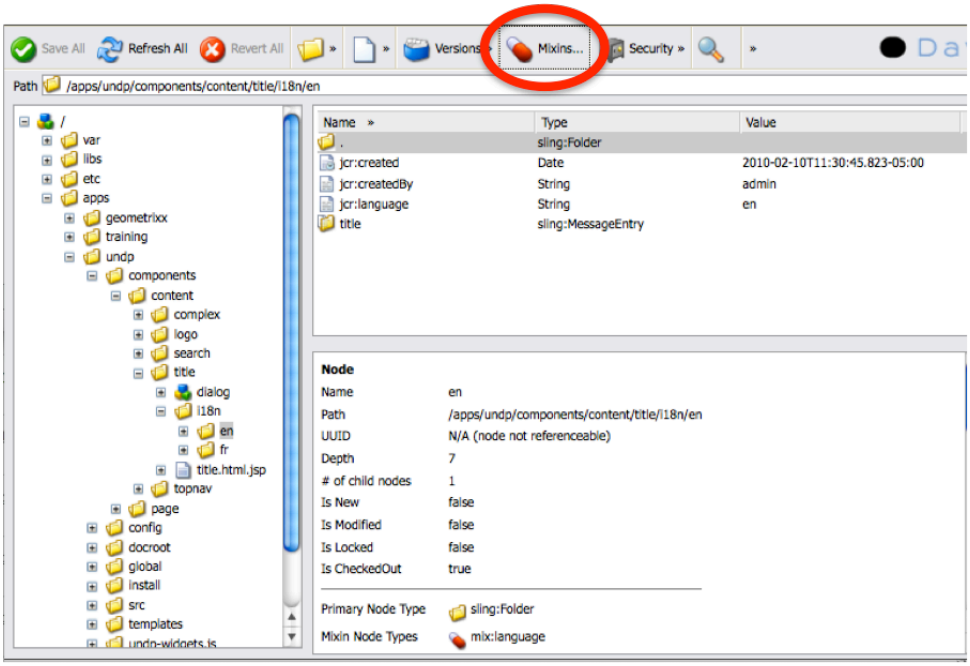
1. Create an **i18n** node (nodeType sling:Folder) under the root node of the title Component.

- e.g. /apps/training/components/content/title

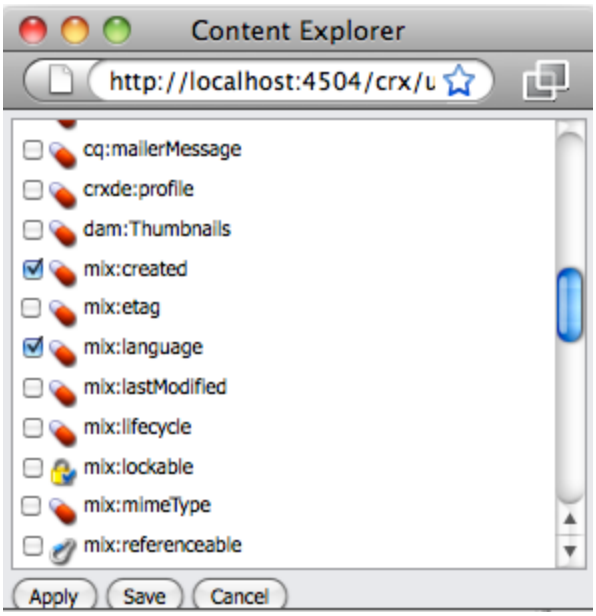
Copyright 2010, Day Software AG, Switzerland

2. Create an **en** node (nodeType sling:Folder) under the newly created *i18n* node.

3. Open a Content Explorer window and assign the **mixin**, mix:language to the newly created *en* node



choose mixin

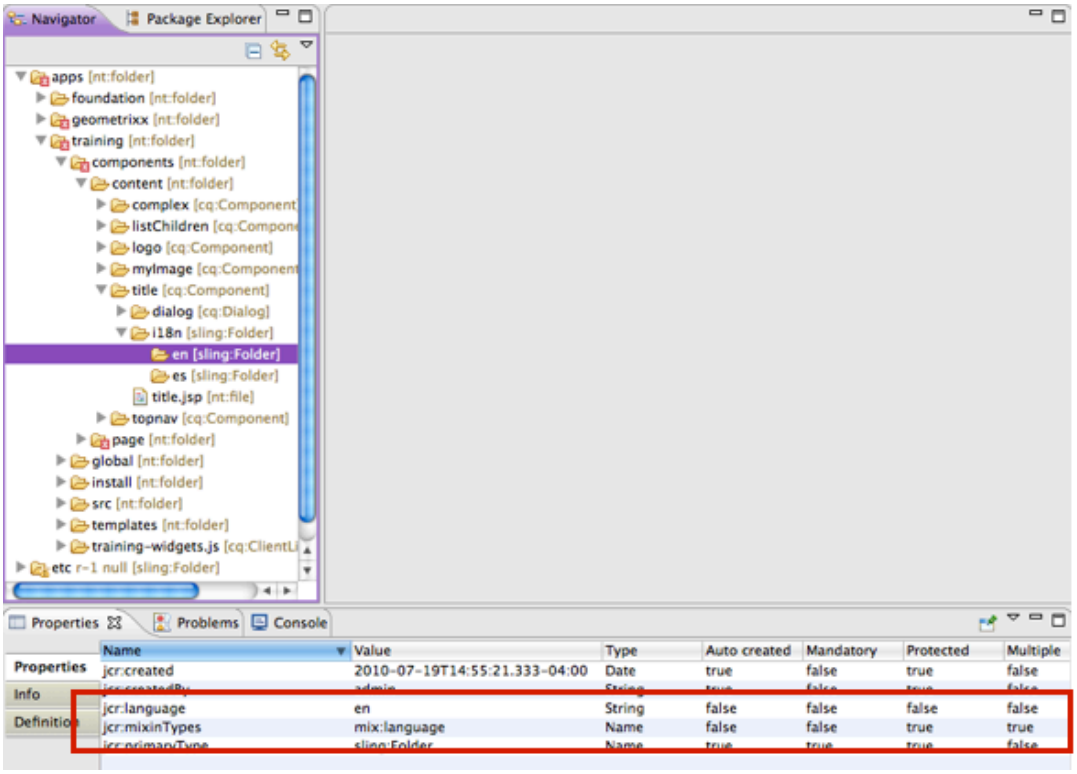


Copyright 2010, Day Software AG, Switzerland

4. Return to CRXDE. Refresh the *en* node.

5. Set the following property on the newly created *en* node:

- the property that identifies the language using the ISO language code
  - Name = jcr:language
  - Type = String
  - Value = en



6. Create a **fieldLabel** node (nodeType sling:MessageEntry) under the newly created *en* node.

7. . Assign the following properties to the newly created *fieldLabel* node:

- the property that identifies the message key
  - Name = sling:key
  - Type = String
  - Value = i18n-title

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- the property that identifies the message
  - Name = sling:message
  - Type = String
  - Value = Enter Title Here (i18n style)

8. Change the **fieldLabel** property (Value = i18n-title) on the title Component's Dialog *title* node.

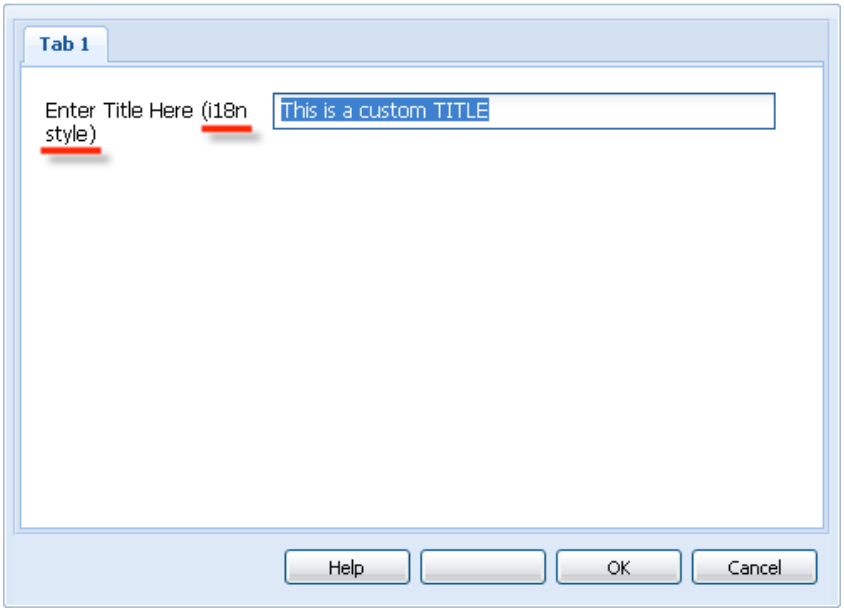
- i.e. ***<path-to-component>/dialog/items/items/tab1/items/title***

9. Repeat steps 2-8 for an additional language. (e.g. Spanish or French)

- ISO code for Spanish = es
- sling:key value = i18n-title
- sling:message value = Titulo

10. Test your script/Dialog by requesting a Page in CQ5 Siteadmin that implements this Training title Component - then invoke the Dialog to observe the changes.

- If successful, you should see Dialog output similar to the image below.



Dialog view of title component - i18n message

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully applied an i18n message to the title Component's Dialog. Since the content author's default/current language preference is English, the en message you defined earlier should appear. If you were to add an additional i18n message, for example in French, and then changed the content author's language preference to fr, the French message you define for the i18n-title key should appear. It might be of best interest to you to try this now.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create & Register a Widget

Goal

The following instructions explain how to create and register a CQ Widget. This will allow you to apply custom Widgets to your Dialogs and Design Dialogs. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed title Component

Why do I need custom Widgets?

Actually, you may never need to create a custom Widget. The CQ and ExtJS Widgets provided to you OOTB cover a large array of potential user input use-cases and are highly configurable. It is Day's strong recommendation you thoroughly study available Widgets and potential configurations via the CQ-WIDGETS-API (i.e. <usb>/distribution/APIs/cq-widgets-api).

The primary objectives for this exercise are:

- Create a JS library
- Extend an existing xtype/Widget
- Register the new xtype/Widget
- Use the xtype/Widget in a Dialog

How to create a JS library for new xtypes/Widgets:

1. Create a **training-widgets.js** node (nodeType *cq:ClientLibraryFolder*) under the root node of the project.

- e.g. /apps/training

Copyright 2010, Day Software AG, Switzerland

2. Assign the following properties to the newly created *training-widgets.js* node:

- the property that identifies the category these custom Widgets will be referenced by
  - Name = categories
  - Type = String[]
  - Value = training.widgets
- the property that defines the component this resource is based on
  - Name = sling:resourceType
  - Type = String
  - Value = widgets/clientlib

3. Create a **files** node (nodeType sling:Folder) under the newly created *training-widgets.js* node.

Congratulations! You have successfully created a JS library for custom xtypes/Widgets. When you wish to add custom Widgets, all you need to do is drop it in the newly created files folder.

Before you can use any custom xtypes/Widgets, you must first register the library in which they exist, which occurs during the WCM initialization process (i.e. init.jsp). To register your JS library you will need to modify the /libs/wcm/core/components/init/init.jsp. Since modifying anything under /libs is strongly not recommended, you will need to create a copy of the original init.jsp for your custom project/modification. In addition, since you will be using a custom init.jsp, you will need to modify the head.jsp that references the init.jsp. The head.jsp that you are currently using is in the Foundation page component because you defined the Foundation page component as the superType of the "page" Component you created earlier.

How to register your newly created JS library using CRXDE:

1.Copy the file */libs/wcm/core/components/init/init.jsp*. Paste *init.jsp* into the */apps/<application name>/global* folder.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

2. Open the file **init.jsp**, and enter some HTML and JSP code, similar to below – then Save.

init.jsp

```
<%@include file="/libs/foundation/global.jsp" %><%
%><%@page import="com.day.cq.wcm.api.WCMMode,
    com.day.cq.widget.HtmlLibraryManager" %><%
if (WCMMode.fromRequest(request) != WCMMode.DISABLED) {
    HtmlLibraryManager htmlMgr = sling.getService(HtmlLibraryManager.class);
    if (htmlMgr != null) {
        htmlMgr.writeCssInclude(slingRequest, out, "cq.wcm.edit");
        out.flush();
        htmlMgr.writeJsInclude(slingRequest, out, "cq.wcm.edit",
"training.widgets");
    }
    String dlgPath = null;
    if (editContext != null && editContext.getComponent() != null) {
        dlgPath = editContext.getComponent().getDialogPath();
    }
    %>
    <script type="text/javascript" >
        var fct = function() {
            CQ.WCM.launchSidekick("<%= currentPage.getPath() %>", {
                propsDialog: "<%= dlgPath == null ? "" : dlgPath %>",
                locked: <%= currentPage.isLocked() %>
            });
        };
        window.setTimeout(fct, 1);
    </script><%
}
%>
```

3. Copy the file */libs/foundation/components/page/head.jsp* – then paste to the Training contentpage "page" Component.

- e.g. /apps/training/components/page/contentpage

4. Open the file **head.jsp**, and enter some HTML and JSP code, similar to below – the Save.

head.jsp

```
<%@include file="/libs/foundation/global.jsp" %><%
%><%@ page import="com.day.cq.commons.Doctype" %><%
```


Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

```
    String xs = Doctype.isXHTML(request) ? "/" : "";
%><head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"<%=xs
%>>
    <meta http-equiv="keywords" content="<%= WCMUtils.getKeywords
(currentPage) %>"<%=xs%>>
    <cq:include script="/apps/training/global/init.jsp"/>
    <cq:include script="stats.jsp"/>
    <% currentDesign.writeCssIncludes(pageContext); %>
    <title><%= currentPage.getTitle() == null ? currentPage.getName() :
currentPage.getTitle() %></title>
</head>
```

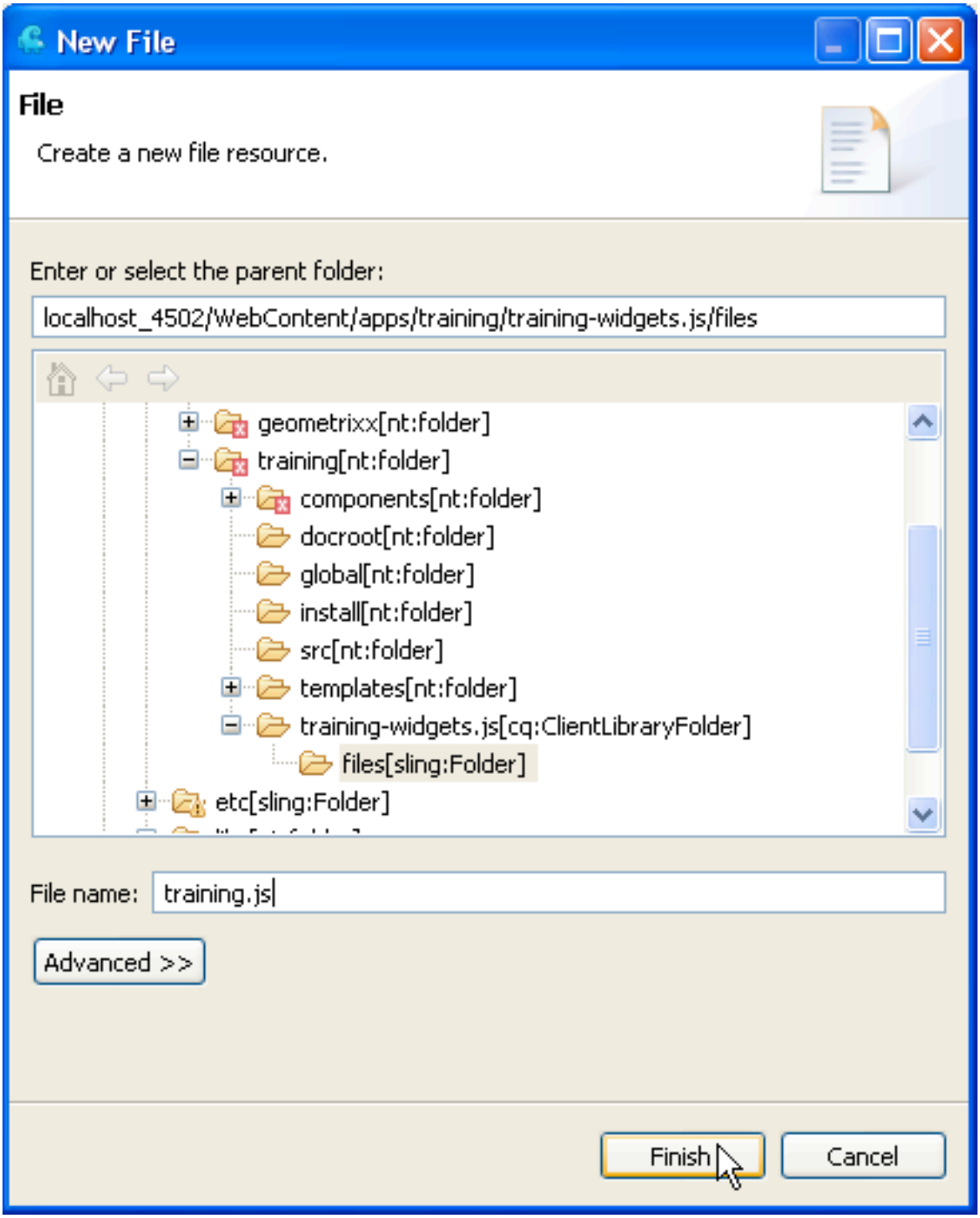
Congratulations! You have successfully registered a JS library for custom xtypes/Widgets. Again, when you wish to add custom Widgets, all you need to do is drop it in the files folder of a registered JS library.

Finally, we will create a new xtype/Widget by extending an existing xtype/Widget. ExtJS is a large, powerful, and somewhat complicated JS framework. If possible, it may benefit you to attend ExtJS training.

Copyright 2010, Day Software AG, Switzerland

How to create and test a custom xtype/Widget:

1. Create a new file in the folder */apps/training/training-widgets.js/files*.



CRXDE new file dialog

Copyright 2010, Day Software AG, Switzerland

2. Open the file **training.js**, and enter some JS code, similar to below - then **Save**.

```
// Create the namespace
Training = {};
// Create a new class based on existing CompositeField
Training.Selection = CQ.Ext.extend(CQ.form.CompositeField, {
    text: "default text",
    constructor : function(config){
        if (config.text != null) this.text = config.text;
        var defaults = {
                height: "auto",
                border: false,
                style: "padding:0;margin-bottom:0;",
                layoutConfig: {
                    labelSeparator: CQ.themes.Dialog.LABEL_SEPARATOR
                },
                defaults: {
                    msgTarget: CQ.themes.Dialog.MSG_TARGET
                }
        };
        CQ.Util.applyDefaults(config, defaults);
        Training.Selection.superclass.constructor.call(this, config);
        this.selectionForm = new CQ.Ext.form.TimeField({
            name: this.name,
            hideLabel: true,
            anchor: "100%",
            minValue: '8:00am',
            maxValue: '6:00pm'
        });
        this.add(this.selectionForm);
    },
    processRecord: function(record, path){
        this.selectionForm.setValue(record.get(this.getName()));
    }
});
CQ.Ext.reg("trainingSelection", Training.Selection);
```

3. Create a **training** node (nodeType cq:Widget) under the Training title Component's Dialog *tab1* widget collection.

- e.g. /apps/training/components/content/title/dialog/items/items/tab1/items

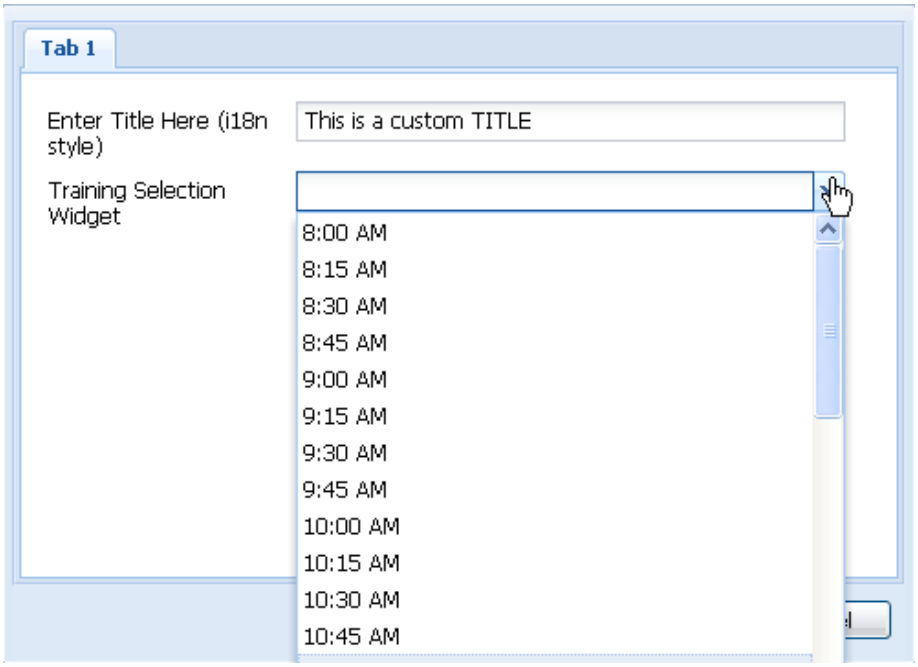
Copyright 2010, Day Software AG, Switzerland

4. Assign the following properties to the newly created *training* node:

- the property that will define where the content is stored
  - Name = name
  - Type = String
  - Value = ./training
- the property that will define the Widget type
  - Name = xtype
  - Type = String
  - Value = trainingSelection
- the property that will define the label applied to the Widget
  - Name = fieldLabel
  - Type = String
  - Value = Training Selection Widget

5. Test your script/Dialog by requesting a Page in CQ5 Siteadmin that implements this Training xtype/Widget (i.e. the title Component) - then double-click the "title" content to invoke the Dialog.

- If successful, you should see the custom Dialog xtype, similar to the image below.



Dialog of title component with new xtype

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully created and applied a custom xtype/Widget. This is a significant step in your development capabilities, as you are now able to create custom CQ xtypes/Widgets. To fully complete this exercise, you would output/use the user's selection in manner deemed appropriate.

Copyright 2010, Day Software AG, Switzerland

Day 4
EXERCISE - Create and Consume an OSGi Bundle

Goal

The following instructions explain how to create and consume an OSGi Bundle. This will allow you to create and use custom Java classes in your JSP scripts, allowing for more traditional Java development and library reuse. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A running CRXDE instance
- A completed title Component

What exactly is an OSGi Bundle?

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.

This architecture allows you to extend Sling with application specific modules. Sling, and therefore CQ5, uses the Apache Felix implementation of OSGi and is based on the OSGi Service Platform Release 4 Version 4.2 Specifications. They are both collections of OSGi bundles running within an OSGi framework.

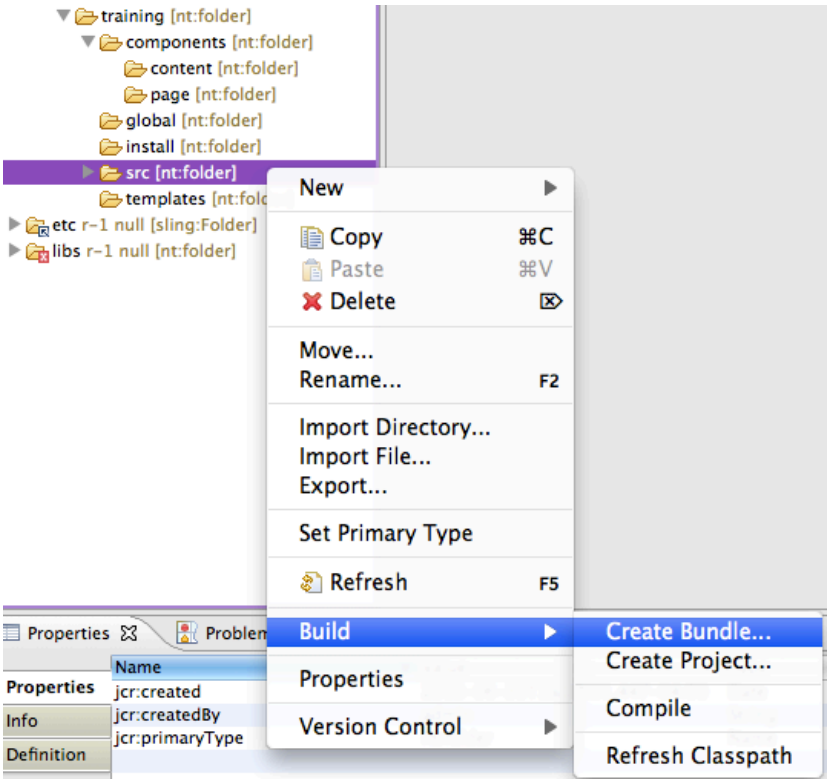
Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

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

How to create and consume an OSGi bundle:

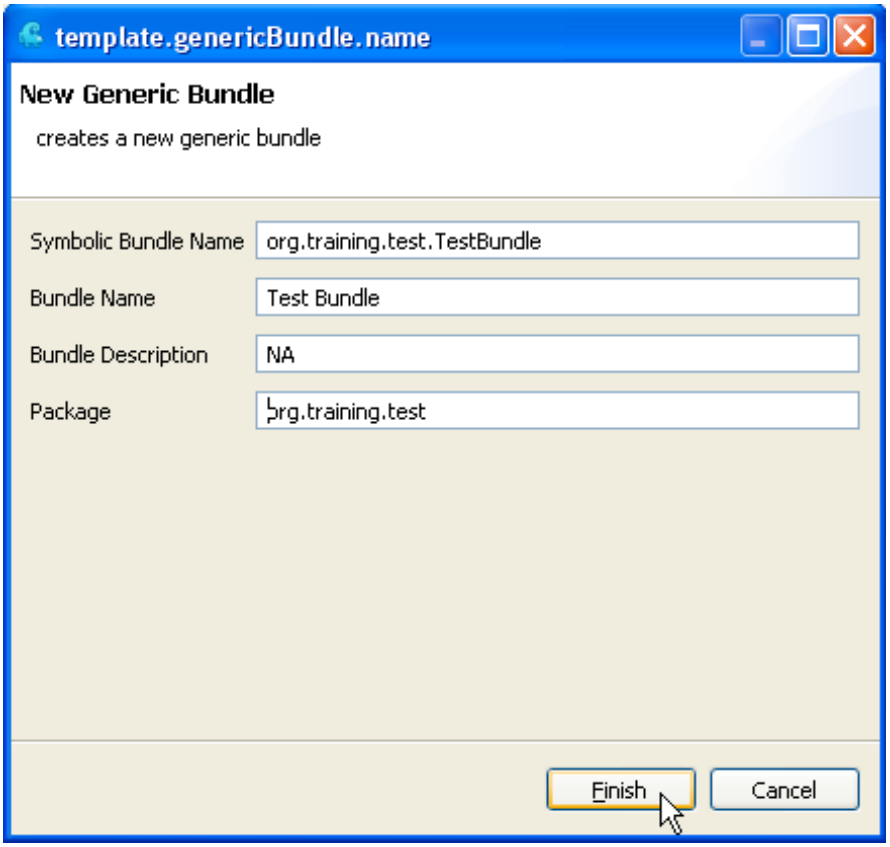
1. Select and right-click the Training projects OSGi bundle source directory (i.e. /apps/training/src) - then select **Build .. Create Bundle**.



CRXDE Build Create Bundle

Copyright 2010, Day Software AG, Switzerland

2. Enter the "Symbolic Bundle Name", "Bundle Name", "Bundle Description", and "Package" - then click Finish.



CRXDE new bundle dialog

3. Open the newly created bundle's file org.training.test.TestBundle.bnd - verify that the text -SNAPSHOT has been added to the Bundle-Version.

- this ensures the bundle is automatically installed and started in the OSGI container every time the bundle is built

```
Export-Package: *
Import-Package: *
Private-Package: *
# Include-Resource:
Bundle-Name: Test Bundle
Bundle-Description: NA
Bundle-SymbolicName: org.training.test.TestBundle
Bundle-Version: 1.0.0-SNAPSHOT
Bundle-Activator: org.training.test.Activator
```

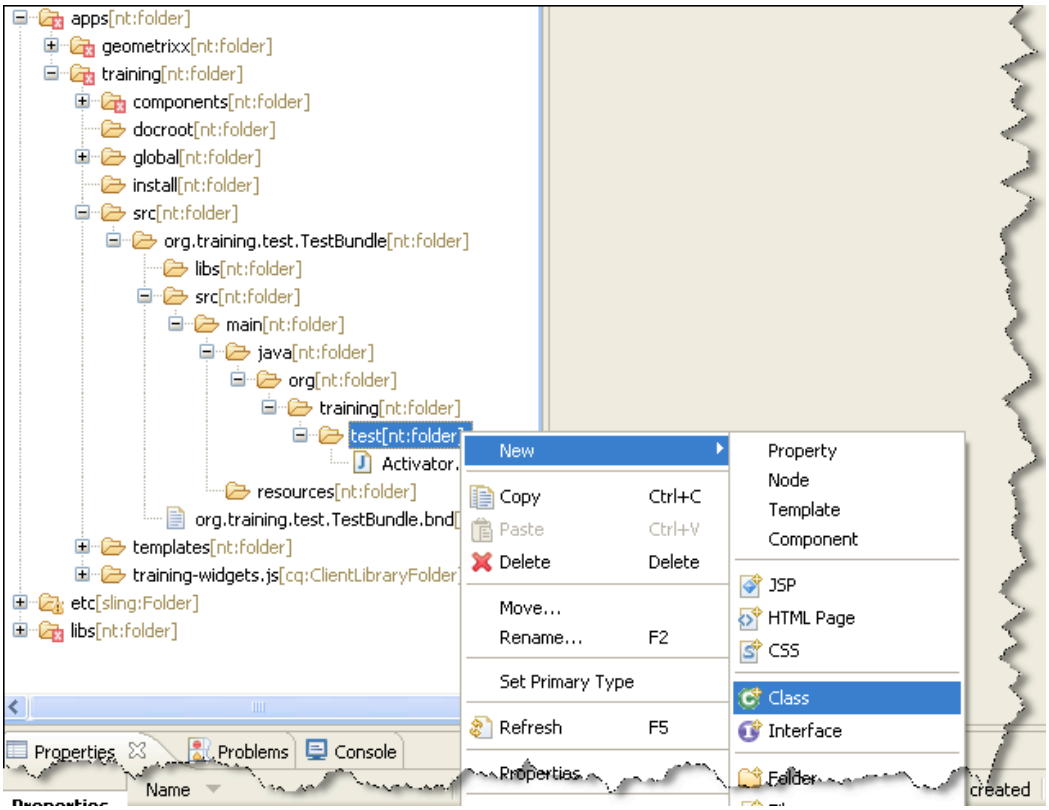

Copyright 2010, Day Software AG, Switzerland

NOTE

The Bundle wizard creates the following elements:

- The node org.training.test.TestBundle of type nt:folder – it is the bundle container node
- The file org.training.test.TestBundle.bnd – it acts as deployment descriptor for your bundle and consists of a set of headers
- The folder structures:
  - src/main/java/org/training/test – it will contain the packages and the Java classes
  - src/main/resources – it will contain the resources used within the bundle
- The Activator.java file – it is the optional listener class to be notified of bundle start and stop events

4. Create a **new Java class** in the newly created bundle source directory (i.e. src/main/java/org/training/test).

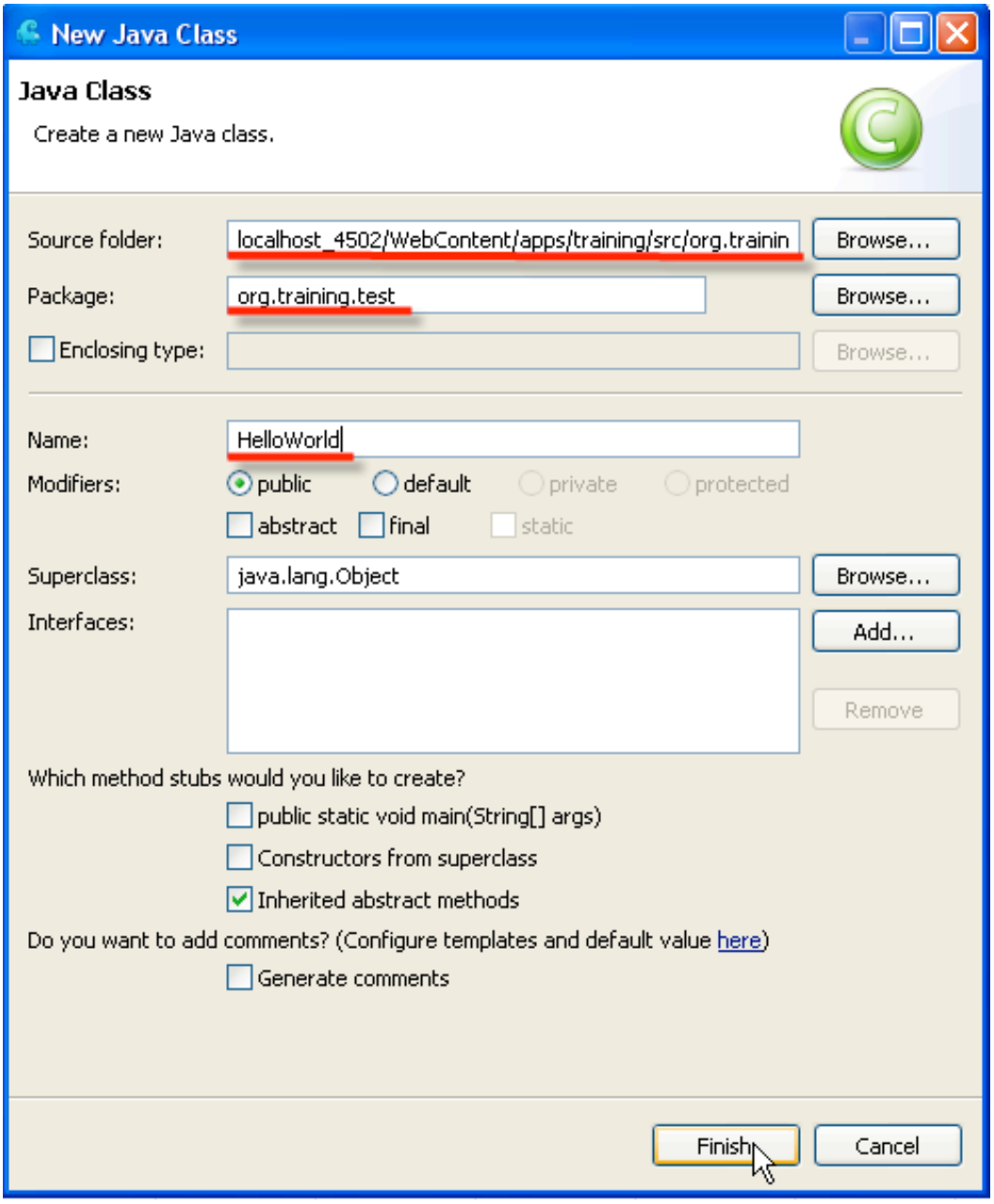


CRXDE create a new java class

5. Enter the "Source folder", "Package", and "Name".

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- Source folder = localhost_4502/WebContent/apps/training/src/org.training.test.TestBundle/src/main/java
- Package = org.training.test
- Name = HelloWorld



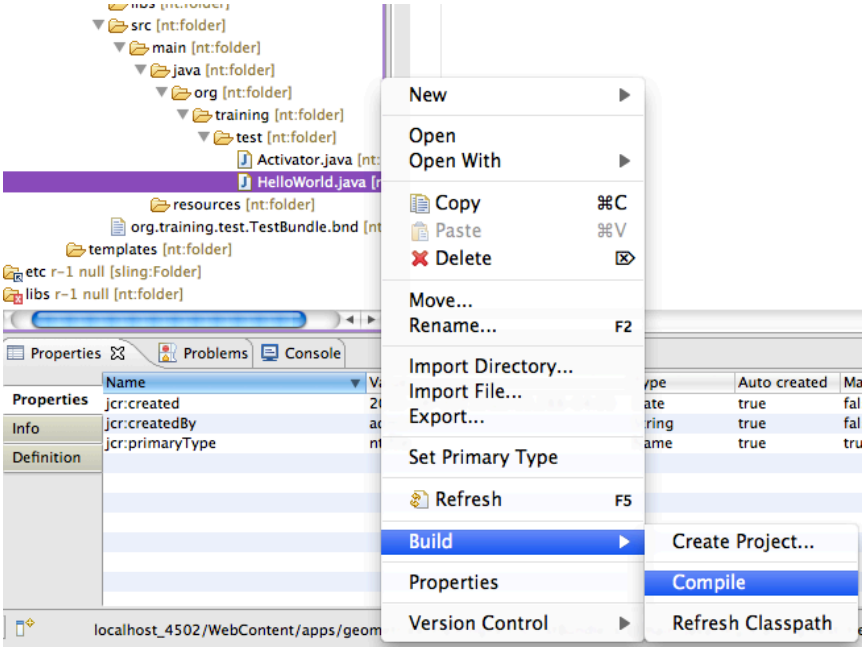
CRXDE new java class dialog

6. Open the newly created Java class file ***HelloWorld.java***, and enter some Java code, similar to below – then **Save**.

Copyright 2010, Day Software AG, Switzerland

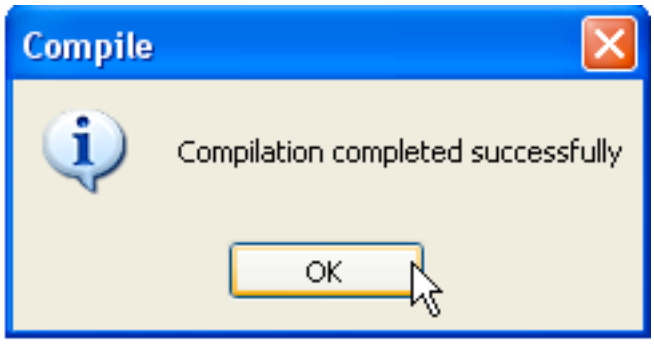
```
package org.training.test;
public class HelloWorld {
      public String getMessage() {
            return "Hello World !!!";
      }
}
```

7. Select and right-click the ***HelloWorld.java*** file – then select **Build .. Compile.**



CRXDE compile Java class

- You should get a Compilation completed successfully message dialog.

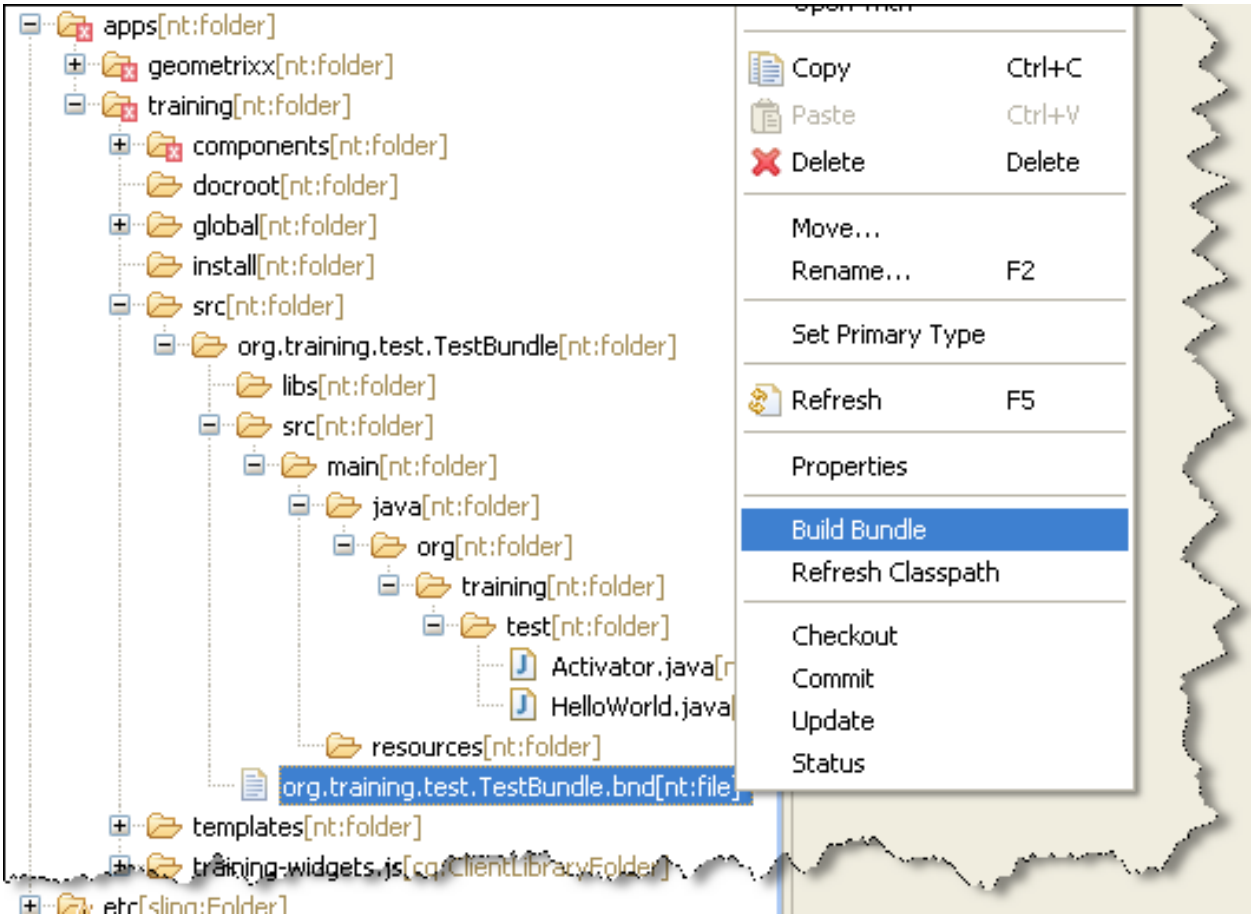


CRXDE Java class compilation successful

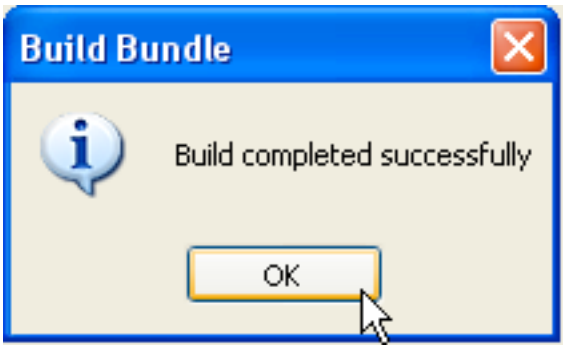
Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

8. Select and right-click the ***org.training.test.TestBundle.bnd*** file - then select **Build Bundle.**

- You should get a Build completed successfully message dialog.



CRXDE build bundle



CRXDE bundle build successful

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

9. Open the Training title Component's ***title.jsp***, and enter some HTML and JSP code, similar to below – then Save.

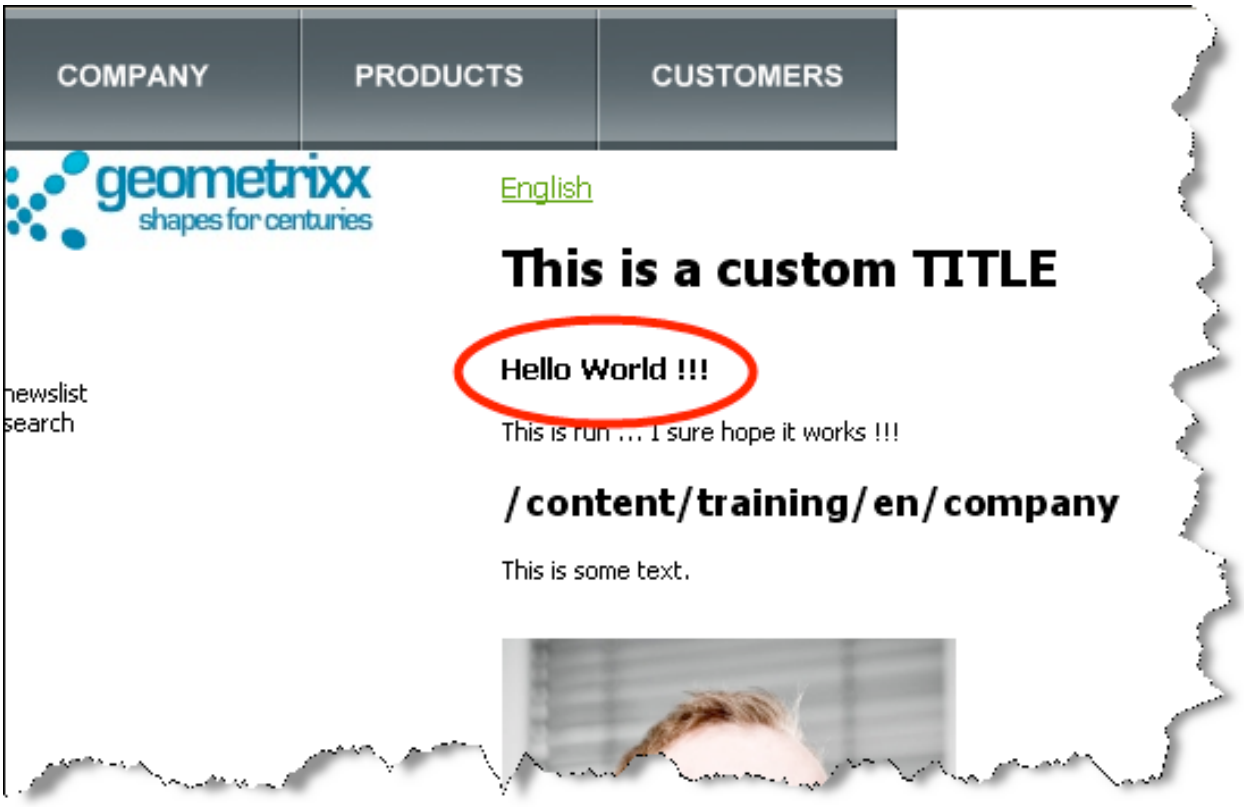
- The focus is to import the newly created Java class and use it

title.jsp

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="org.training.test.HelloWorld" %>
<h1><%= properties.get("title", currentPage.getTitle()) %></h1>
<%
HelloWorld hw = new HelloWorld();
%>
<h3><%= hw.getMessage() %></h3>
```

10. Test your bundle/script by requesting a Page in CQ5 Siteadmin that implements this Training bundle (i.e. the title Component).

- If successful, you should see the title Component output, similar to the image below.



Page view of title component (bundle)

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully created an OSGi bundle, and have consumed the associated Java class. Once again, this is a significant step in your development capabilities, as you are now able put more "heavy logic" type of code into Java classes, where it belongs, in addition to being able to create Java helper classes that can be re-used by your development team.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Examine the Workflow Console

Goal

CQ5 encompasses several applications that are designed to interact and complement each other. In particular, the Workflow Engine can be used in tight conjunction with several of the other applications.

For example, within CQ5, Web Content Management (CQ WCM) is key. This enables you to generate and publish pages to your website. This functionality is often subject to organizational processes, including steps such as approval and sign-off by various participants. These processes can be represented as workflows, which in turn can be defined within CQ, then applied to the appropriate content pages.

The following instructions will increase familiarity with the Workflow Console and working with Workflows. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance

Overview of the main workflow objects

Model

Is made of WorkflowNodes and WorkflowTransitions. The transitions connect the nodes and define the „flow“. The Model has always a start node and an end node.

Workflow models are versioned. Running Workflow Instances keep the initial workflow model version that is set when the workflow is started.

Step

There are different types of workflow steps:

- Participant (User/Group)
- Process (Script, Java method call)

Copyright 2010, Day Software AG, Switzerland

- Container (Sub Workflow)
- OR Split/Join
- AND Split/Join

All the steps share the following common properties: **AutoAdvance** and **Timeout** alerts (scriptable).

Transition
Defines the link between two consecutive steps. It is possible to apply rules to the Transition.

WorkItem
The workItem is the "there is a task identifier" and is put into the respective inbox. A workflow instance can have one or many WorkItems at the same time (depending on the workflow model).

The WorkItem references the workflow instance. In the repository the WorkItem is stored below the workflow instance.

Payload
References the resource that has to be advanced through a workflow.

The payload implementation references a resource in the repository (by either a path or an UUID) or a resource by a URL or by a serialized java object. Referencing a resource in the repository is very flexible and in conjunction with sling very productive: for example the referenced node could be rendered as a form.

Lifecycle
Is created when starting a new workflow (by choosing the respective workflow model and defining the payload) and ends when the end node is processed.

The following actions are possible on a workflow instance:

- Terminate
- Suspend
- Resume
- Restart

Completed and terminated instances are archived.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Inbox
Each logged in user has its own workflow inbox in which the assigned WorkItems are accessible. The WorkItems are assigned either to the user itself or to the group to which he belongs.

Workflow Console

The Workflow console is the centralized location for workflow management in CQ. It can be accessed via the Workflows button on the Welcome page or through the Workflows button in the toolbar on any CQ5 console (for example: Websites, Tools, Tagging).

Within the console there are 5 tabs:

Inbox
Lists workflows awaiting action in your inbox. You can then take action as required.

Models
Lists the workflow models currently available. Here you can create, edit or delete workflow models.

Instances
Shows you details of workflow instances which are currently active. These instances are also version dependent.

Archive
Enables you to access details of workflow instances which have terminated, for whatever reason.

Launcher
Allows you to define a workflow to be launched if a specific node has been updated.

Starting a Workflow

There are three methods of starting a workflow:

- Workflow Console
- SiteAdmin Console (Websites tab)

Copyright 2010, Day Software AG, Switzerland

- Sidekick

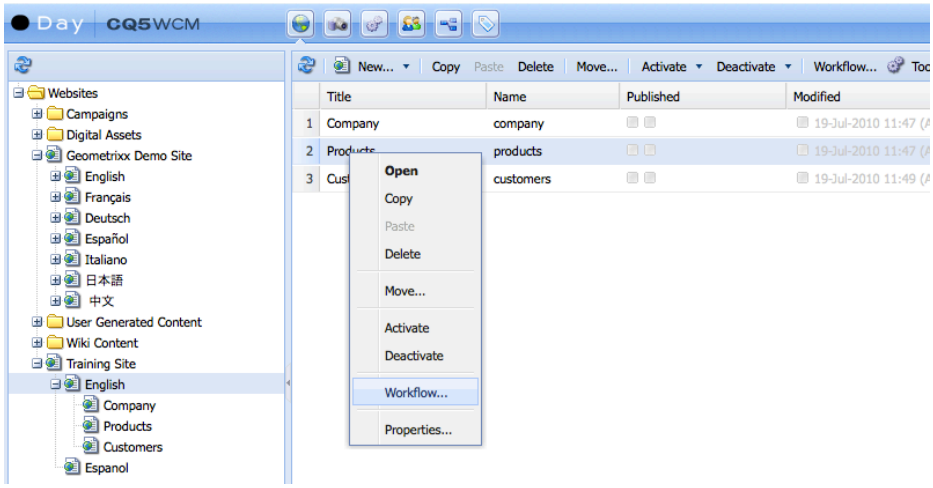
NOTE

The current version of the workflow model is assigned to the payload; if the main copy of the workflow is updated later, the changes will have no impact on the currently running instance.

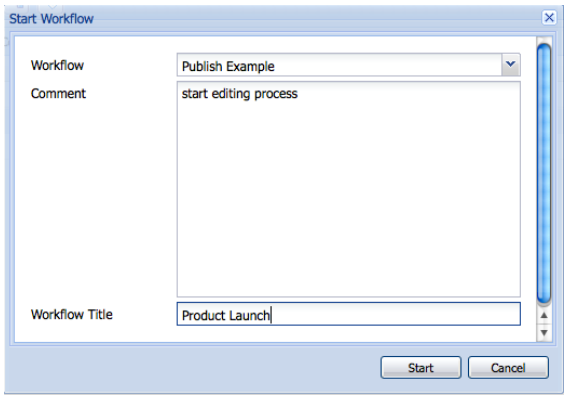
We will be starting the workflow from the SiteAdmin console.

1. Open the SiteAdmin console, either by clicking on the Websites button from the Welcome page or by clicking on the globe icon at the bottom of your sidekick.

2. Right-click on one of your pages to open the context menu. Select **Workflow...**

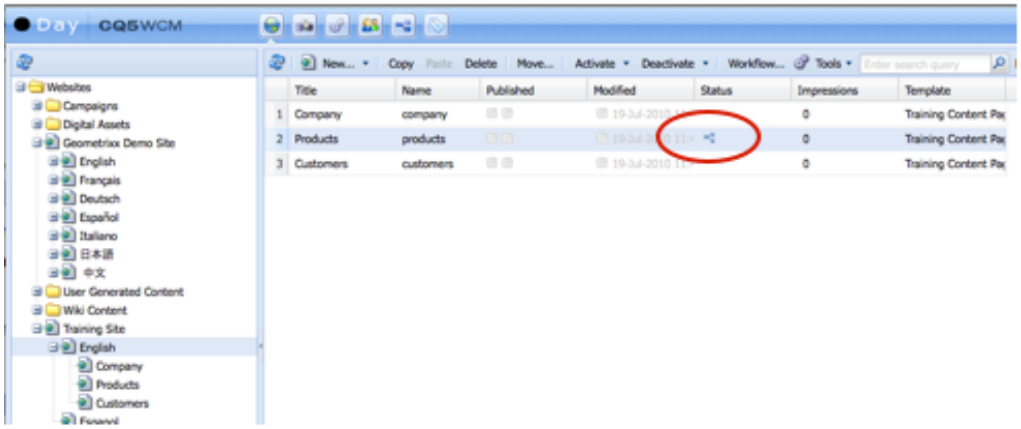


3. Select the **Publish Example** workflow and fill in any desired optional information and click **Start**.

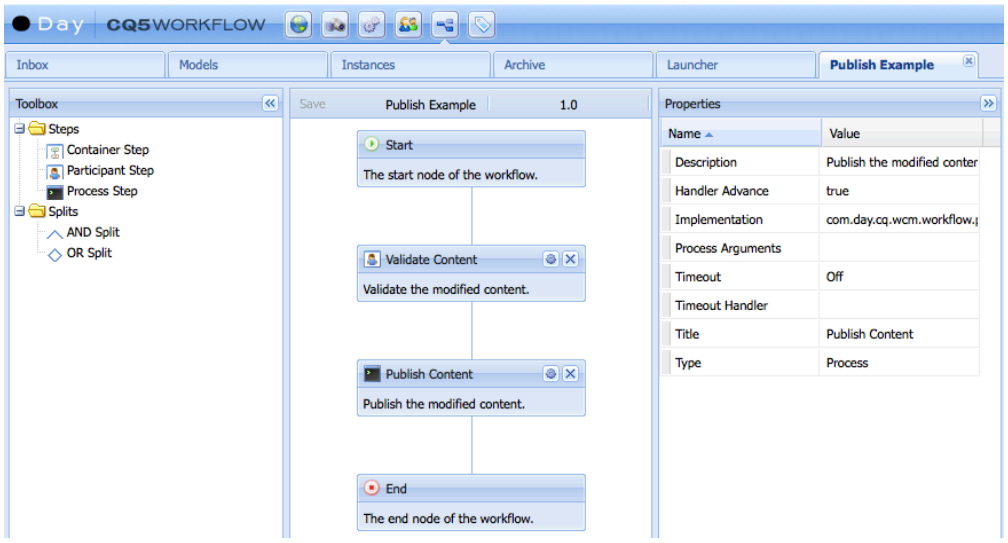


Copyright 2010, Day Software AG, Switzerland

4. You will notice that the workflow icon shows up in the SiteAdmin console next to the page you chose.



5. Change to the Workflow Console by clicking on the workflow button in the toolbar. Select the **Models** tab. Open the **Publish Example** workflow by double-clicking the workflow model entry.

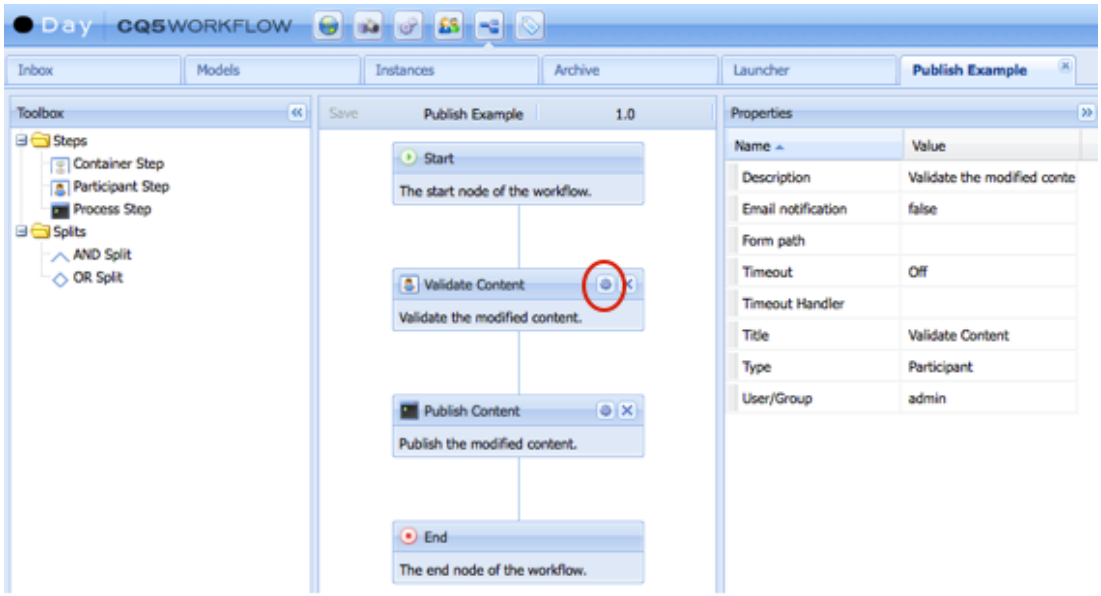


You will notice that the **Publish Example** workflow has 2 steps: **Validate Content** and **Publish Content**.

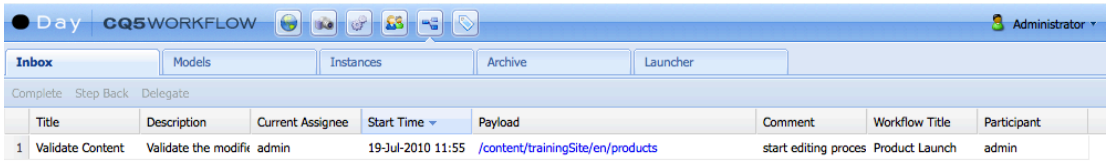
The **Validate Content** step is a Participant step assigned to the user admin. The **Publish Content** step is a Process step with the Implementation Property set to ActivatePageProcess.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

You can validate the values of the step properties by clicking on the cog in the step and then inspecting the values from the properties list on the right.

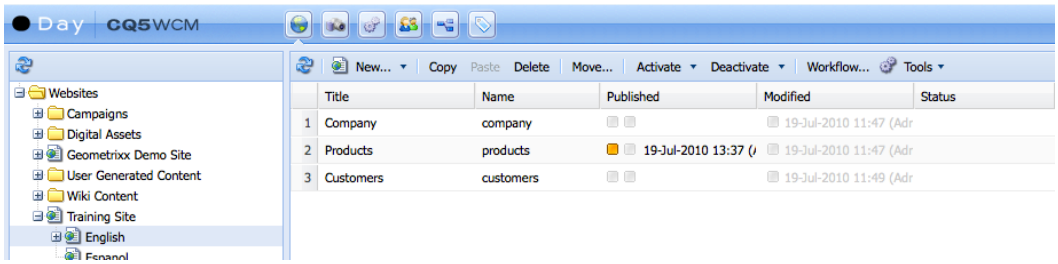


6. Click on the **Inbox** tab to change context. The Inbox will show the WorkItem entry for the page you put into workflow.



7. Select the WorkItem **Validate Content** and click **Complete**. Notice that the WorkItem disappears from the Inbox. You will also note that the page is now no longer in workflow (the workflow icon is gone from the status column) and the page has been published or is pending publication (depending on whether you have a CQ5 Publish instance running).

Copyright 2010, Day Software AG, Switzerland



Congratulations! You have learned about the Workflow Console. Also, you have successfully placed a page in workflow and moved that page through the workflow steps to the the end of the workflow. In the next exercise, you will use this knowledge to help you create a custom process step and define a workflow to use that custom process step.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

EXERCISE - Create a Workflow Implementation Step

Goal

A workflow is made of steps. The steps that define a workflow are either participant steps or process steps. Participant steps require manual intervention by a person to advance the workflow. Process steps, on the other hand, are automatic actions that are executed by the system if certain specified conditions are met.

CQ provides a number of predefined process steps that perform common actions, which an administrators can use when building a new workflow. Custom process steps can also be added for tasks not covered by the built-in steps.

Process steps, also called automated steps, can be defined by using either an ECMA script or a service (a Java class in a bundle). Services can be developed to listen to special workflow events and perform tasks according to the business logic.

The following instructions explain how to create a workflow custom process step using a Java class. To successfully complete and understand these instructions, you will need:

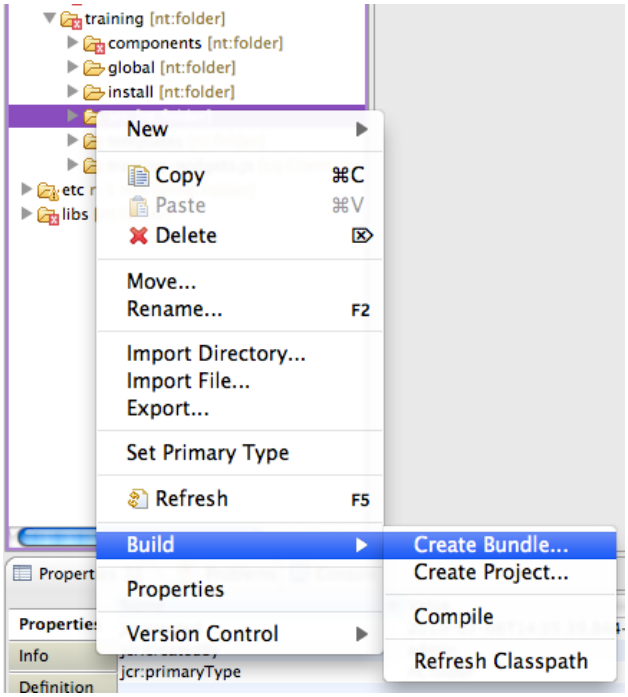
- A running CQ5 Author instance
- A completed Training project with appropriate extensions

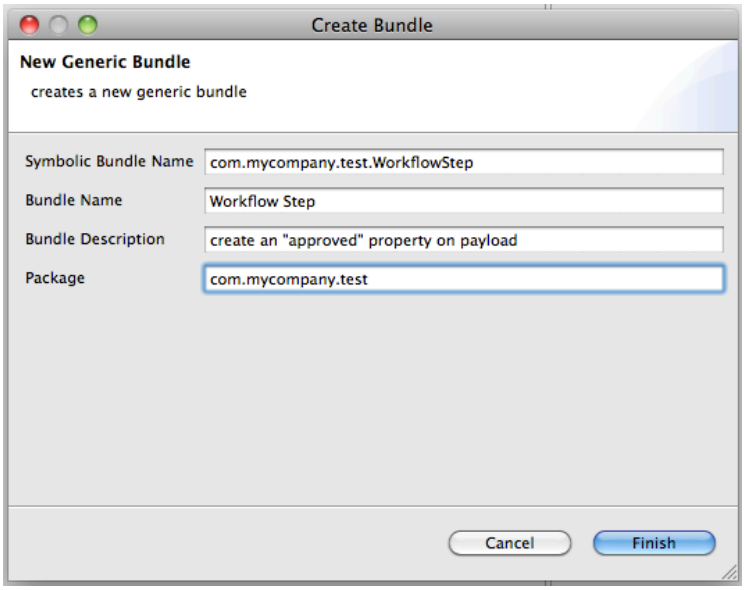
Defining a process step: with a Java class

To define a process step as an OSGI service component (Java bundle):

1. Create the bundle to hold the Java class using **Build..Create Bundle.**

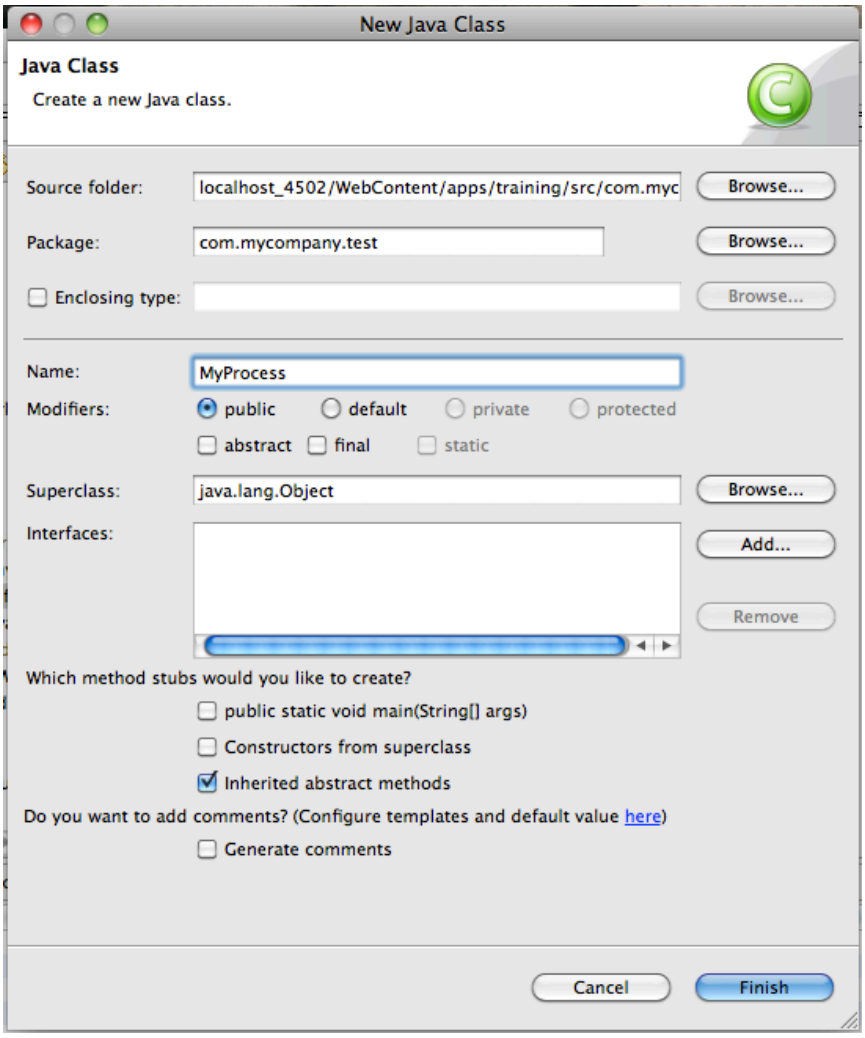
Copyright 2010, Day Software AG, Switzerland





2. Create a Java class to implement the creation of the **approved** property.

Copyright 2010, Day Software AG, Switzerland



3. Open the newly created Java class file ***MyProcess.java***, and enter some Java code, similar to below – then **Save**.

```
package com.mycompany.test;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.day.cq.workflow.WorkflowSession;
import com.day.cq.workflow.exec.JavaProcessExt;
import com.day.cq.workflow.exec.WorkItem;
import com.day.cq.workflow.exec.WorkflowData;
import javax.jcr.Node;
```


Copyright 2010, Day Software AG, Switzerland

```
/**
 * <code>MyProcess</code> sets the "approve" property to the payload
 *
 * @scr.component metatype="false"
 * @scr.service
 */

public class MyProcess implements JavaProcessExt {

    /** Default log. */
    protected final Logger log = LoggerFactory.getLogger(MyProcess.class);

    public void execute(WorkItem item, WorkflowSession session, String
args[]) throws Exception {
        WorkflowData workflowData = item.getWorkflowData();
        if (workflowData.getPayloadType() == "JCR_PATH") {
            String path = workflowData.getPayload().toString() + "/
jcr:content";
            Node node = (Node) session.getSession().getItem(path);
            if (node != null) {
                node.setProperty("approved", args[0].equals("true") ?
true : false);
                session.getSession().save();
            }
        }
    }

    public void execute(WorkItem item, WorkflowSession session) throws
Exception {
        String[] args = new String[]{"true"};
        execute(item, session, args);
    }
}
```

NOTE

Pay special attention to the ***@scr.component metatype="false"*** and ***@scr.service*** comments. These two lines are required to ensure that the implementation step appears in the list of implementations.

4. Compile the Java class using **Build..Compile**.

NOTE

The java methods, respectively the classes that implement the executable Java method are registered as OSGI services, enabling you to add methods at anytime during runtime.

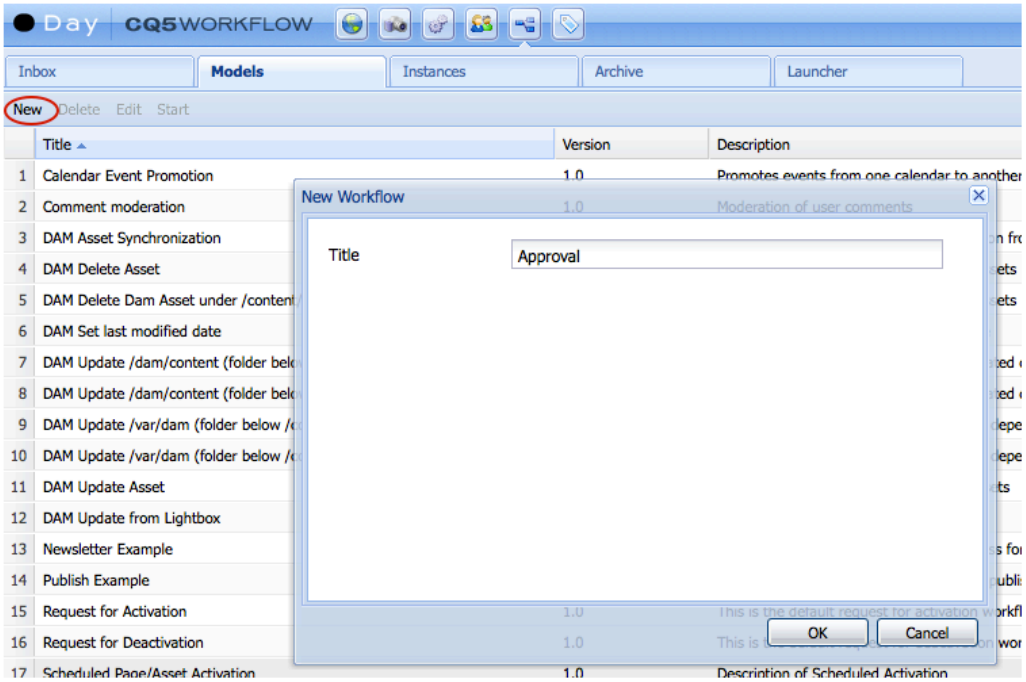
Copyright 2010, Day Software AG, Switzerland

5. Build the Bundle using **Build..Build Bundle**. Once you get the "Build Successful" message, you will notice that the bundle ***.jar*** file shows up in the install directory. The bundle is now uploaded and installed into CQ.

Congratulations! You have created a new Java class that can be used as a custom Implementation in a Process step.

Use the new Process Implementation in a Workflow

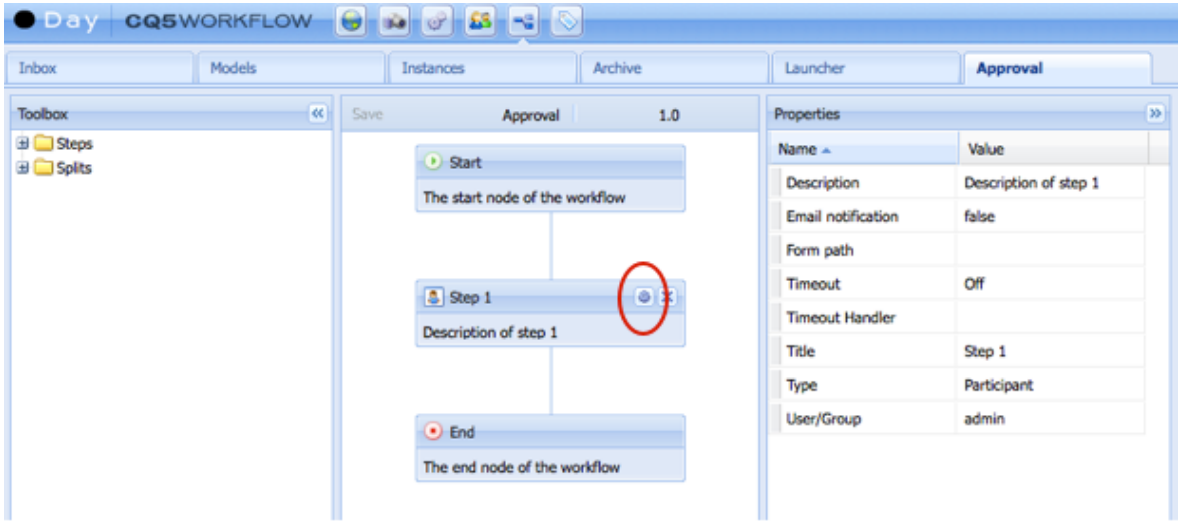
1. Navigate to the Workflow panel. Choose the **Models** tab. Create a workflow by choosing **New** from the toolbar. Enter *Approval* as the Title of the workflow.



2. Open the **Approval** workflow by double-clicking on the Name.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

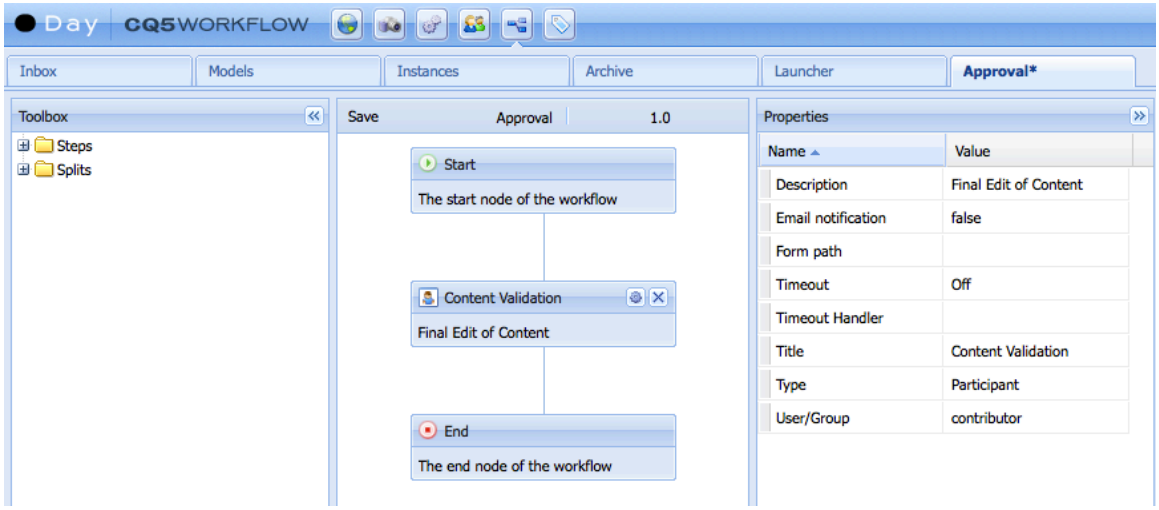
3. You will notice that the new **Approval** workflow model has a Start, End and Step 1.



To edit Step 1, click on the cog  on the step itself.

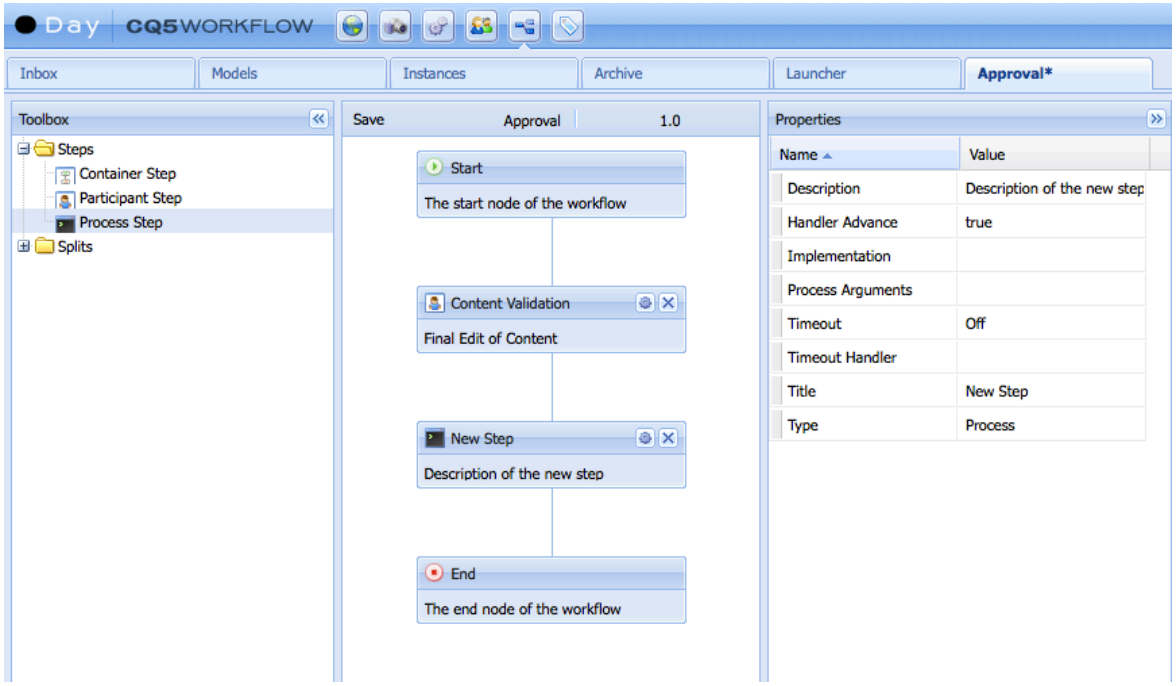
4. Assign the following values for the Step 1 properties:

- Description = Final Edit of Content
- Title = Content Validation
- User/Group = Contributors

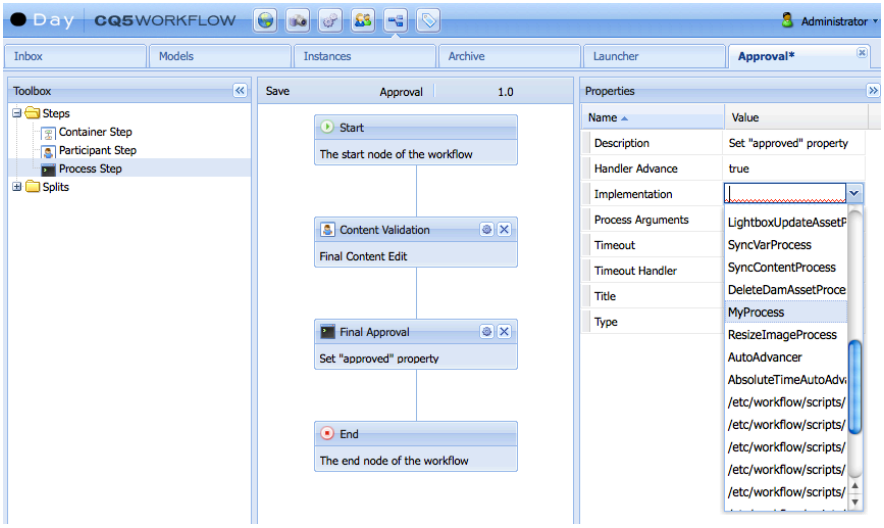


Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

5. Add a Process step to the workflow by dragging-and-dropping the Process icon onto the workflow model.



6. Set the properties of your new Process step:
   - Description = Set “approved” property
   - Implementation = MyProcess
   - Title = Final Approval
   - Process Arguments = true



Copyright 2010, Day Software AG, Switzerland

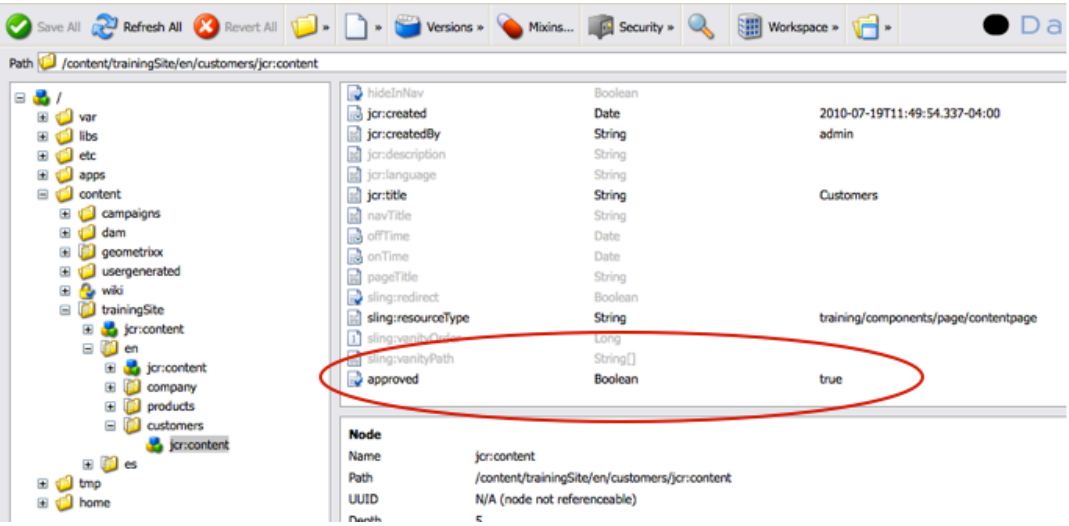
You will see the following value in the Implementation property:

```
com.mycompany.test.MyProcess
```

Click **Save**.

7. Test your new workflow. In the WebSites panel, select any page. Bring up the right context menu and choose **Workflow**... Select the new Approval workflow and click **Start**.

8. Monitor the progress of your workflow from the inbox and move the page through the workflow. Check for the ***approved*** property on the page once the workflow has completed.



Congratulations! You have successfully created a custom Implementation for a workflow process step. You can use this same methodology for any custom Implementation you need to create for your workflows.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Create & Download a CQ Package

Goal

The following instructions explain how to create a CQ package that will combine all elements of the Training project, minus all jpegs. This is a good example of packaging application content, which you could then distribute to team members for review. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A completed Training project with appropriate extensions

Why do I need CQ packages?

Packages can include content and project-related data. A package is a zip file that contains the content in the form of a file-system serialization (called "vault" serialization) that represents the content from the repository as an easy-to-use-and-edit representation of files and folders.

Additionally, it contains vault meta information, including a filter definition, and import configuration information. Additional content properties can be included in the package, such as a description, a visual image, or an icon. These properties are for the content package consumer for informational purposes only.

You can perform the following actions with packages:

- Create new packages
- Modify existing packages
- Build packages
- Upload packages
- Install packages
- Download packages from the package share library
- Download packages from CQ to a local machine
- Apply package filters
- View package information

Copyright 2010, Day Software AG, Switzerland

There are multiple ways to manage content packages:

- CRX Package Manager – using the direct CRX interface
- CQ Package Manager
- cURL actions – command line option that allows package actions to be automated

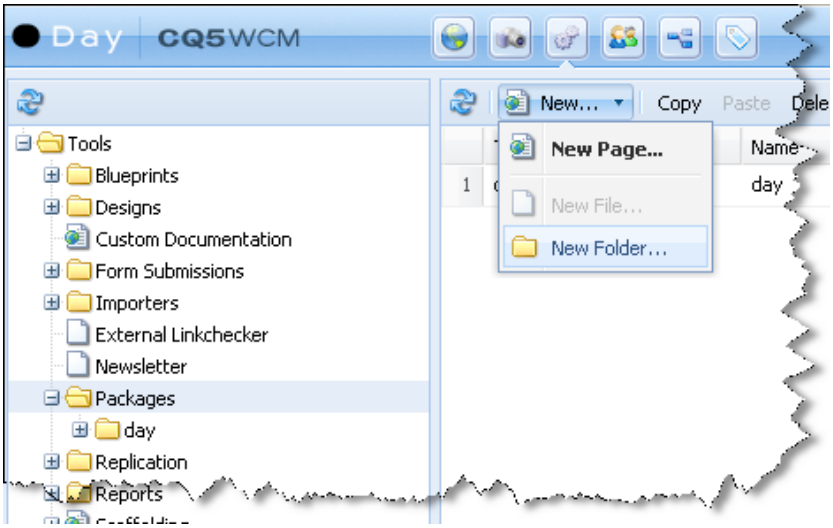
In this exercise, we will be using the CQ Package Manager. You should take some time to explore the documentation for the CRX and cURL methods of managing packages.

How to create, build, and download a CQ package in the "Tools" section of CQ5:

1. Navigate to the Tools pane.

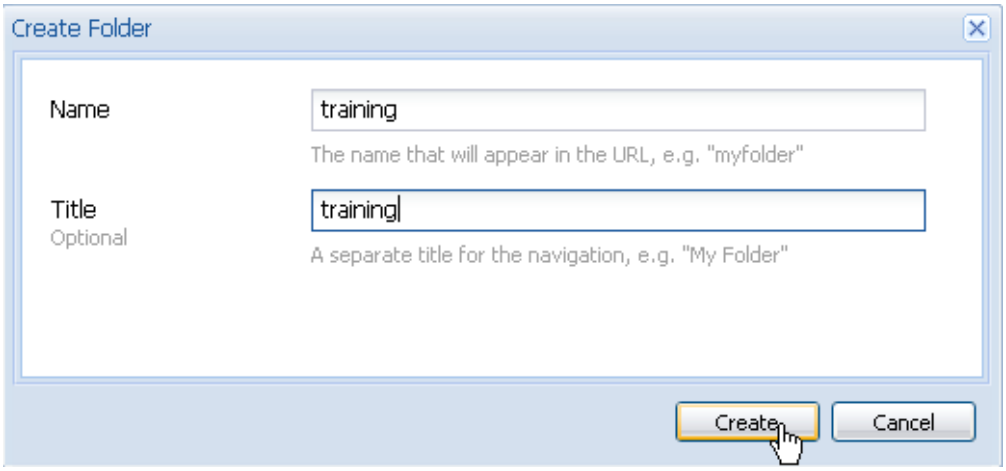
2.Select the **Packages** Folder.

3.Create a new training folder.



CQ5 packages - new folder

Copyright 2010, Day Software AG, Switzerland

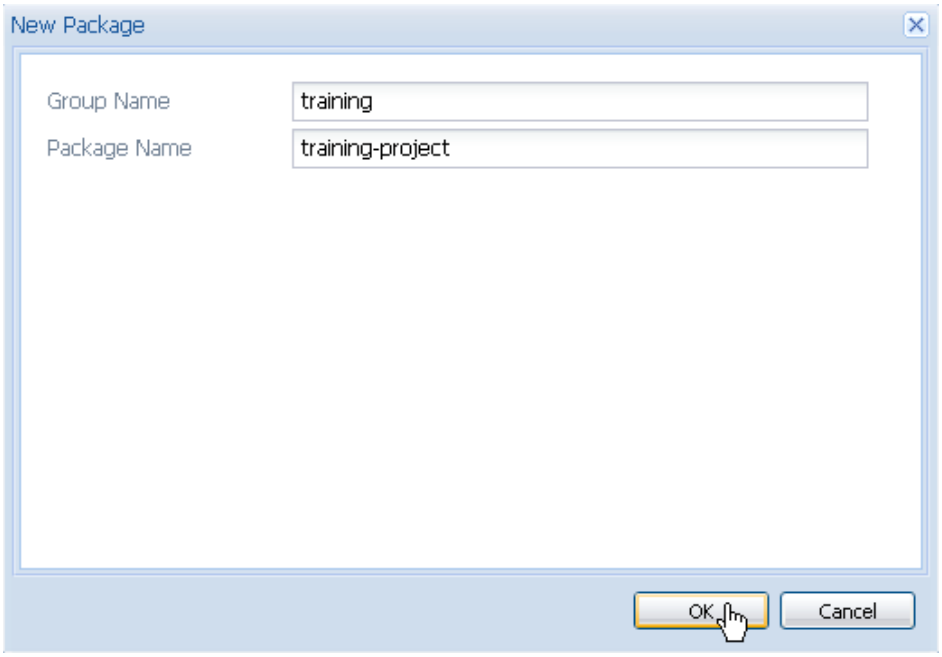


New folder dialog

4. Double-click the newly created training folder.

5. Select **Create Package**.

6. Enter the package "Group Name" (training) and "Package Name" (training-project).

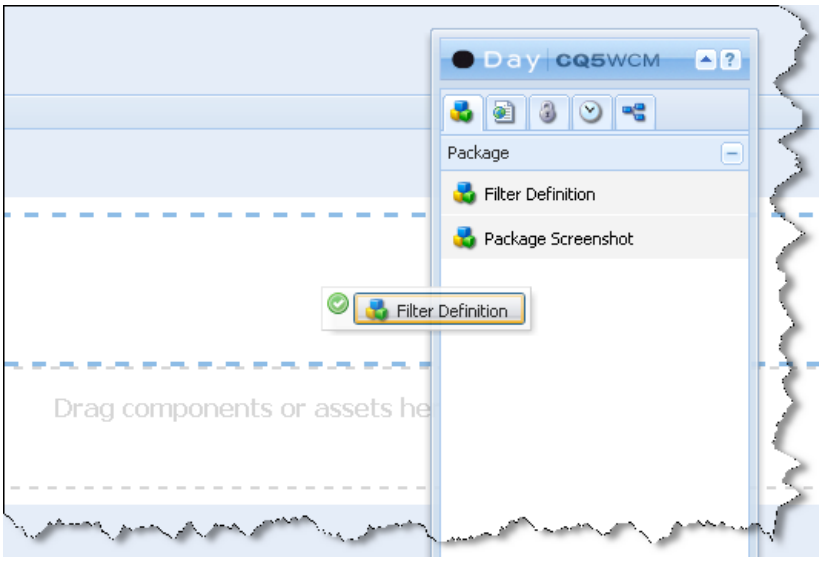


CQ new package dialog

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

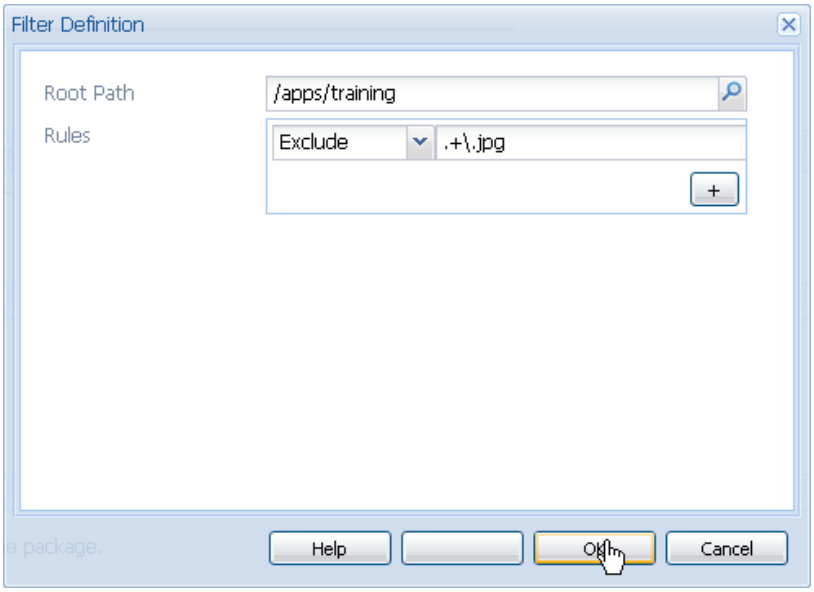
7. Select the training-project package.

8. Add the Component **Filter Definition** to the paragraph system Component - then open (e.g. double-click).



Page view of component addition

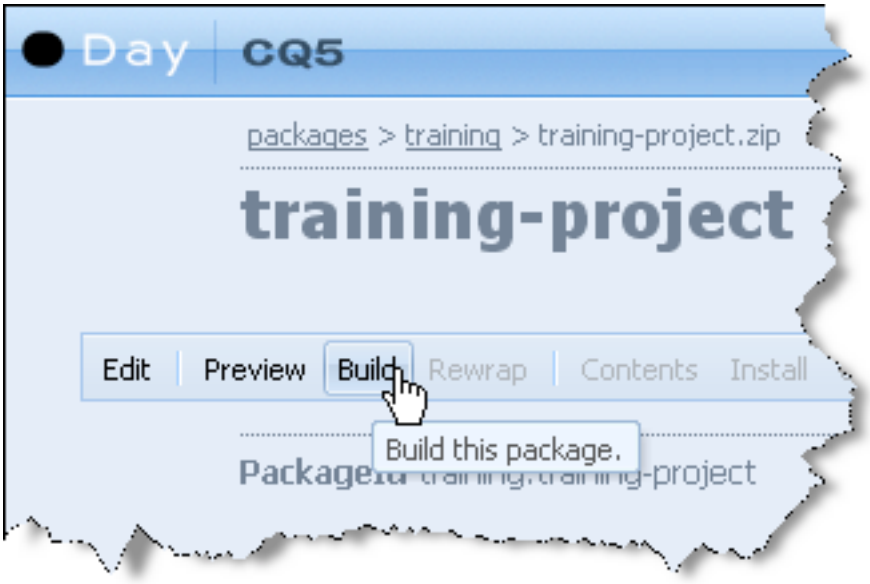
9. Enter the "**Root Path**" (/apps/training) and a "**Rule**" that excludes all jpegs (Exclude => .+\.jpg) - then select OK.



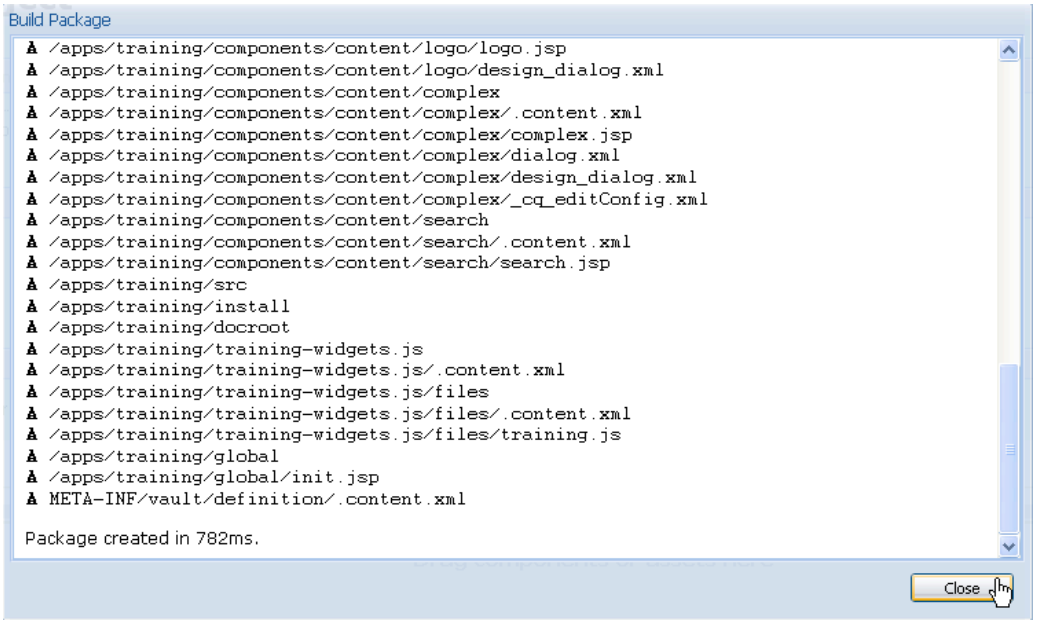
Filter definition dialog

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

10. Select Build to build the package.

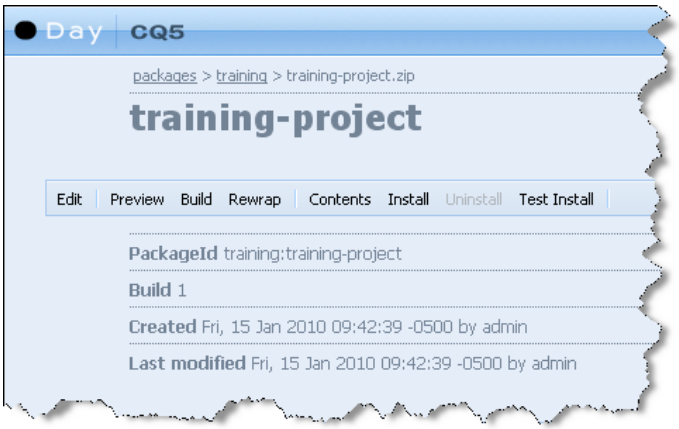


Package build selection



Package build output

Copyright 2010, Day Software AG, Switzerland



Package build information

11. Download the package by entering the URL of the package's ZIP in your Web browser's address bar.

- e.g. http://localhost:4502/etc/packages/training/training-project.zip

Congratulations! You have successfully created a package, added a rule to the filter definition, built the package, and have downloaded the package, which you can now share with your CQ development team.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Find Slow Responses

Goal

The following instructions explain how to monitor Page response times, in addition to finding slow Page responses. This will allow you to create a more robust/scalable CQ application, even with limited hardware. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance
- A Day provided rlog.jar tool

What performance optimization concepts should I consider?

A key issue is the time your Web site takes to respond to visitor requests. Although this value will vary for each request, an average target value can be defined. Once this value is proven to be both achievable and maintainable, it can be used to monitor the performance of the Web site and indicate the development of potential problems.

The response times you will be aiming for will be different on the author and publish environments, reflecting the different characteristics of the target audience:

Author Environment

This environment is used by authors entering, and updating content, so it must cater for a small number of users who each generate a high number of performance intensive requests when updating content pages and the individual elements on those pages.

Publish Environment

This environment contains content which you make available to your users, where the number of requests is even greater and the speed is just as vital, but since the nature of the requests is less dynamic, additional performance enhancing mechanisms can be leveraged, such as that the content is cached or load-balancing is applied.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Performance Optimization Methodology

A performance optimization methodology for CQ projects can be summed up to five very simple rules that can be followed to avoid performance issues from the start. These rules, to a large degree, apply to Web projects in general, and are relevant to project managers and system administrators to ensure that their projects will not face performance challenges when launch time comes.

Planning for Optimization

Around 10% of the project effort should be planned for the performance optimization phase. Of course, the actual performance optimization requirements will depend on the level of complexity of a project and the experience of the development team. While your project may ultimately not require all of the allocated time, it is good practice to always plan for performance optimization in that suggested range.

Whenever possible, a project should first be soft-launched to a limited audience in order to gather real-life experience and perform further optimizations, without the additional pressure that follows a full announcement.

Once you are "live", performance optimization is not over. This is the point in time when you experience the "real" load on your system. It is important to plan for additional adjustments after the launch.

Since your system load changes and the performance profiles of your system shifts over time, a performance "tune-up" or "health-check" should be scheduled at 6-12 months intervals.

Simulate Reality

If you go live with a Web site and you find out after the launch that you run into performance issues there is only one reason for that: Your load and performance tests did not simulate reality close enough.

Simulating reality is difficult and how much effort you will reasonably want to invest into getting "real" depends on the nature of your project. "Real" means

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

not just "real code" and "real traffic", but also "real content", especially regarding content size and structure. Keep in mind that your templates may behave completely different depending on the size and structure of the repository.

Establish Solid Goals

The importance of properly establishing performance goals is not to be underestimated. Often, once people are focused on specific performance goals, it is very hard to change these goals afterwards, even if they are based on wild assumptions.

Establishing good, solid performance goals is really one of the trickiest areas. It is often best to collect real life logs and benchmarks from a comparable Web site (for example the new Web site's predecessor).

Stay Relevant

It is important to optimize one bottleneck at a time. If you do things in parallel without validating the impact of the one optimization, you will lose track of which optimization measure actually helped.

Agile Iteration Cycles

Performance tuning is an iterative process that involves, measuring, analysis, optimization and validation until the goal is reached. In order to properly take this aspect into account, implement an agile validation process in the optimization phase rather than a more heavy-weight testing process after each iteration.

This largely means that the developer implementing the optimization should have a quick way to tell if the optimization has already reached the goal, which is valuable information, because when the goal is reached, optimization is over.

Basic Performance Guidelines

Generally speaking, keep your uncached html requests to less than 100ms. More specifically, the following may serve as a guideline:

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- 70% of the requests for pages should be responded to in less than 100ms.
- 25% of the requests for pages should get a response within 100ms-300ms.
- 4% of the requests for pages should get a response within 300ms-500ms.
- 1% of the requests for pages should get a response within 500ms-1000ms.
- No pages should respond slower than 1 second.

The above numbers assume the following conditions:

- measured on publish (no authoring environment and/or CFC overhead)
- measured on the server (no network overhead)
- not cached (no CQ-output cache, no Dispatcher cache)
- only for complex items with many dependencies (HTML, JS, PDF, ...)
- no other load on the system

There are a certain number of issues that frequently contribute to performance issues which mainly revolve around (a) dispatcher caching inefficiency and (b) the use of queries in normal display templates. JVM and OS level tuning usually do not lead to big leaps in performance and should therefore be performed at the very tail end of the optimization cycle.

Your best friends during a usual performance optimization exercise are the request.log, component based timing, and last but not least - a Java profiler.

How to monitor Page response times:

1. Navigate to and open the file request.log located at *<cq-install-dir>/crx-quickstart/logs*.

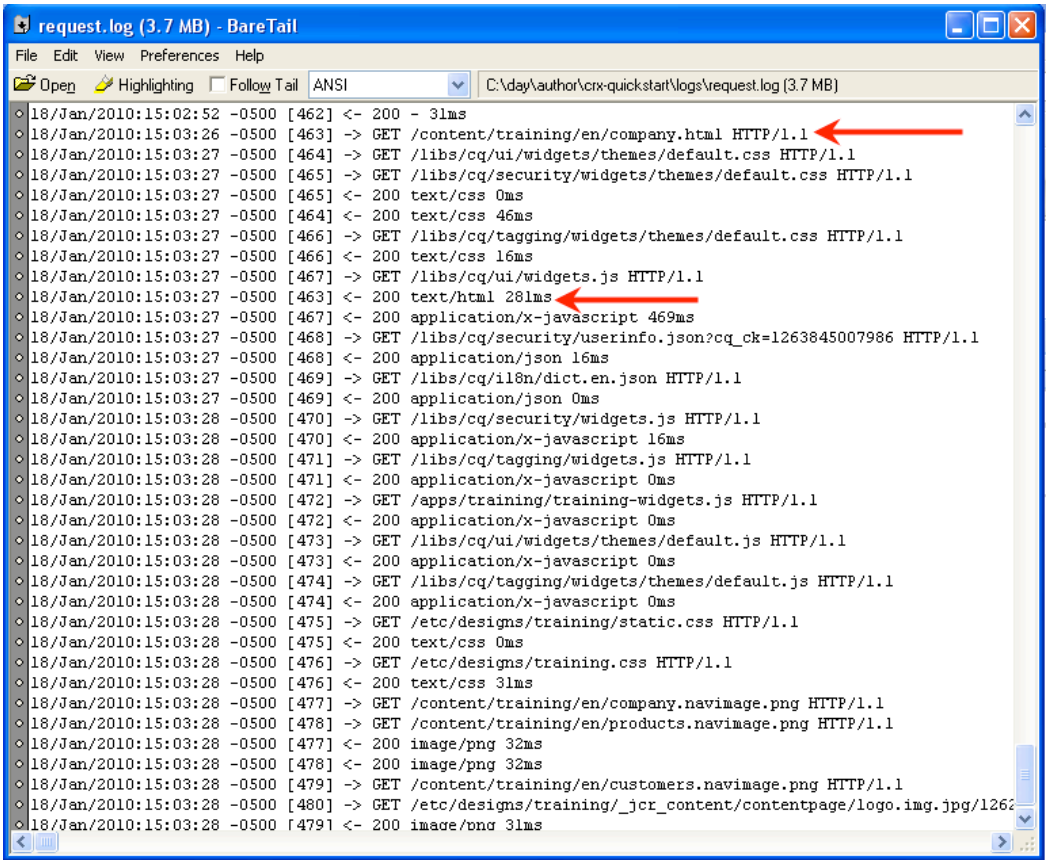
2. Request a Page in author that utillizes your Training Template and Components.

- e.g. /content/trainingSite/en/company

3. Review the response times directly related to the previous step's request.

- A Page request of /content/trainingSite/en/company

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



Request.log response time

NOTE

Though this is clearly the response time of a Page while in author, it is good practice to be able to identify where response times are located (request.log) and how to identify a specific Page request/response in the log file itself. Ideally, this test would occur on the publish instance to get a better idea of its expected behavior in production.

Congratulations! You have successfully reviewed the response time of a Page in CQ using the request.log. Again, this will aid your development in being able to monitor response times of Pages that implement custom Templates and Components, and comparing said time to your project goals.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

How to monitor Component based timing:

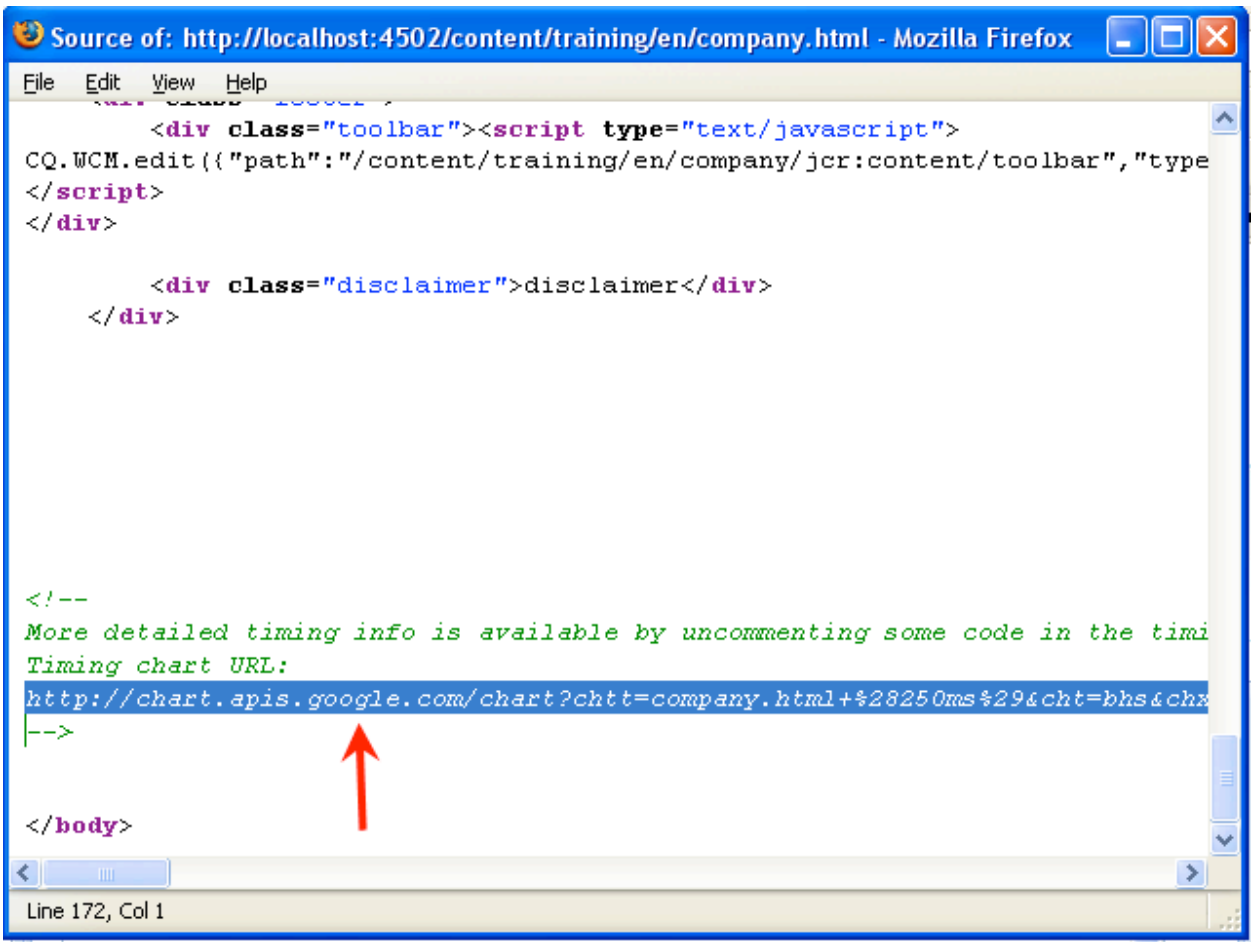
1. Request a Page in author that was created from your Training Template and Components.

- e.g. /content/trainingSite/en/company

2. View the HTML source of the Page requested in step 1.

3. Navigate to and select the "Timing chart URL" located in the HTML source.

- You will find this URL most likely near the bottom of the HTML source, as it is generated by the foundation timing Component

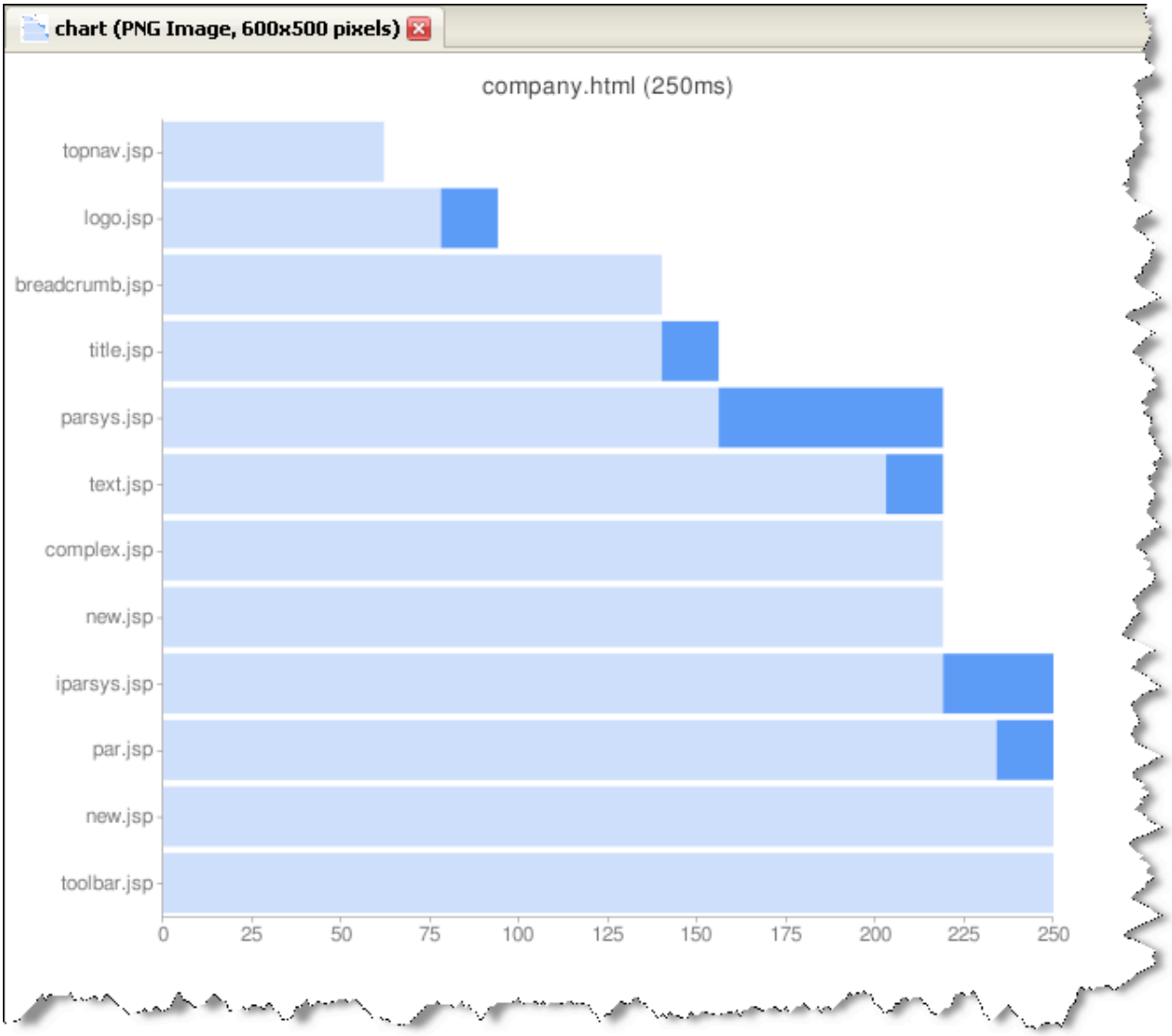


HTML source timing chart url

Copyright 2010, Day Software AG, Switzerland

4. Copy the "**Timing chart URL**" – then paste it in the address bar of your favorite Web browser.

5. Investigate the visual output to identify any Component that may be causing a slow response time.



Timing chart component response times

NOTE

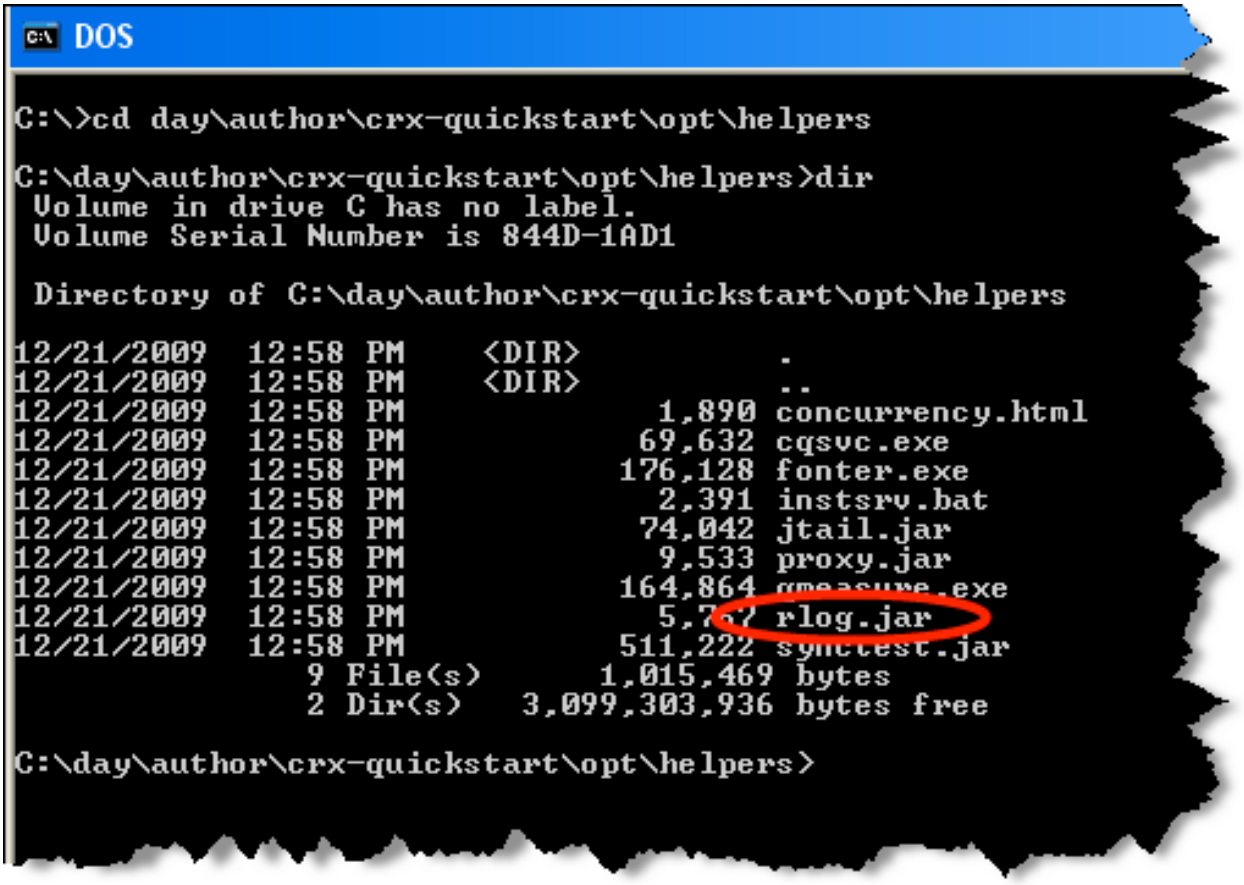
The light blue color identifies when the Component/JSP started. The dark blue color identifies how long the Component/JSP took to respond.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

Congratulations! You have successfully monitored Component based timing using the foundation timing Component. Again, this will aid your development in being able to monitor the response times of specific Components within the context of an actual Page/Template request. Next, we will focus on how to find long lasting request/response pairs.

How to find long lasting request/response pairs:

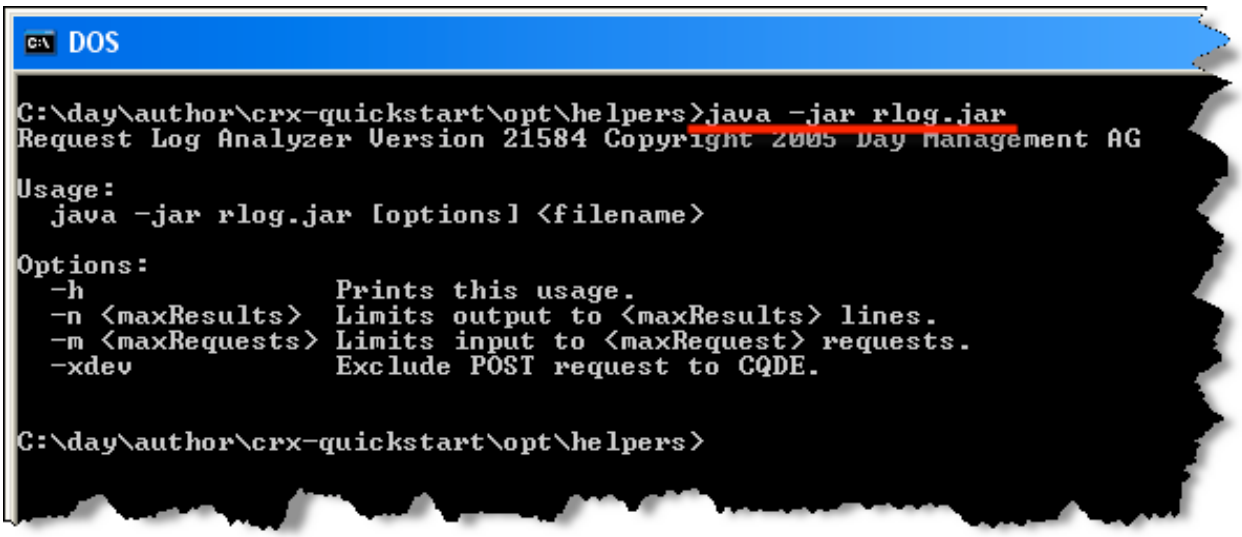
1. Navigate to the helper tool rlog.jar located in <cq-install-dir>/crx-quickstart/opt/helpers using your command line.



DOS location of rlog.jar

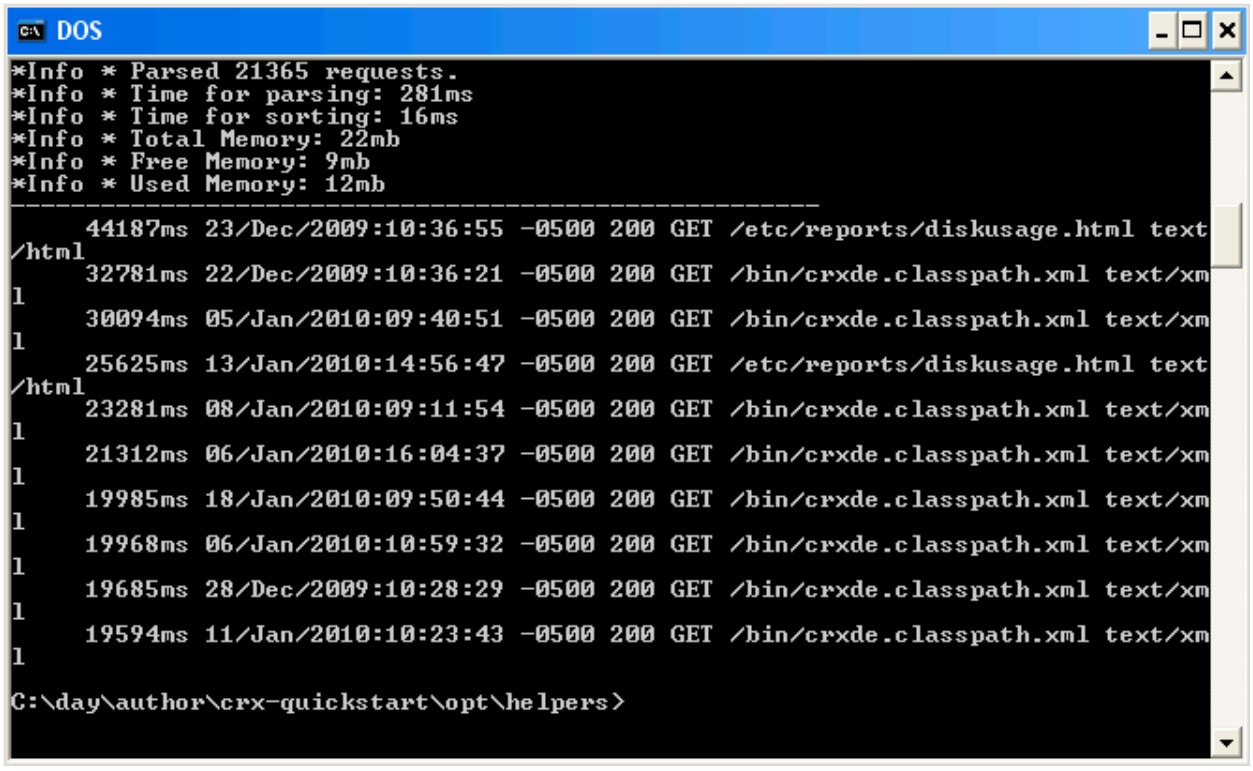
2. Enter the command **java -jar rlog.jar** in your command line to get help concerning possible arguments.

Copyright 2010, Day Software AG, Switzerland



DOS rlog.jar help

3. Enter the command **java -jar rlog.jar -n 10 ../../logs/request.log** in your command line to view the first 10 requests with the longest response duration.



DOS rlog.jar view of 10 longest requests/responses

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

NOTE

Essentially, the rlog.jar tool is parsing the request.log to find and display the 10 longest requests/responses. You can use this information to further investigate what may be the cause of the long response.

Congratulations! You have successfully found and displayed long lasting requests/responses in CQ. Again, this is just one of many tools to help you meet your project's performance goals. Finally, we will view timing statistics for Page rendering.

How to view timing statistics for Page rendering:

1. Request a Page in CQ5 Siteadmin that implements your Training Template and Components.

- e.g. /content/trainingSite/en/company

2. Select Ctrl-Shift-U to view the timing statistics for that Page.

Page timing statistics

Copyright 2010, Day Software AG, Switzerland

Congratulations! You have successfully viewed the timing statistics for a Page. Again, this is to aid you in reviewing the performance of specific Pages, so that you may meet your project's performance goals.

Copyright 2010, Day Software AG, Switzerland

EXERCISE - Change Default Passwords

Goal

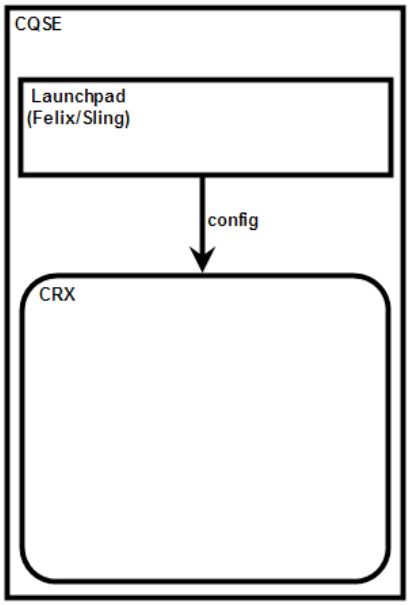
The following instructions explain how to change the default passwords of CQ. This is important because it is part of the security checklist that will ensure your installation cannot be easily infiltrated by hackers. To successfully complete and understand these instructions, you will need:

- A running CQ5 Author instance

What to do about security?

Typically, security tasks are handled by a system administrator. That being said, it is a good idea for you, the developer, to have a basic understanding of what security concerns there are. The primary security concern you will focus on in this exercise is the simple changing of passwords, so that you may setup a team development environment as soon as the class is over.

When considering a standard CQ installation, there are three password changes and one configuration you need to alter. If you consider a standard installation, and the elements involved, it actually becomes quite clear. Reflect on the image below:



CQ standard installation

Copyright 2010, Day Software AG, Switzerland

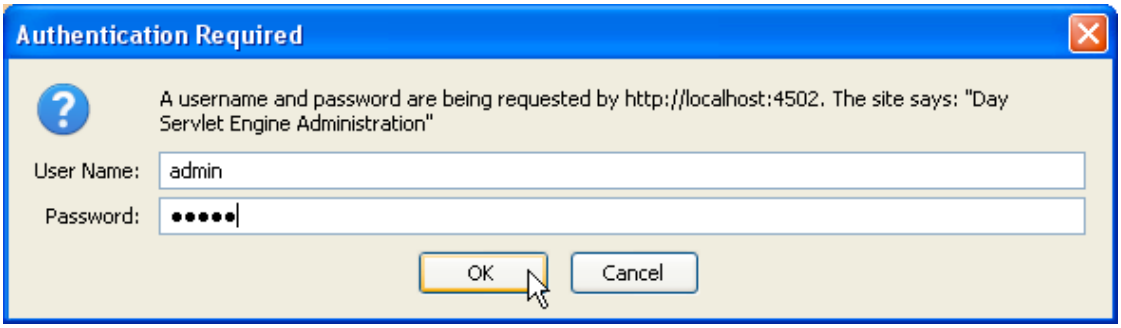
As you can see, there is a Java servlet engine (CQSE), the Launchpad application (Felix/Sling), and the content repository (CRX). All three have their own separate (default) passwords, and thus need to be changed. In addition, the Launchpad application needs to be able to communicate with the content repository. This is a configuration you will have to alter.

How to change the Java servlet engine's (CQSE) default administrative password:

1. Navigate to CQSE's administration application.

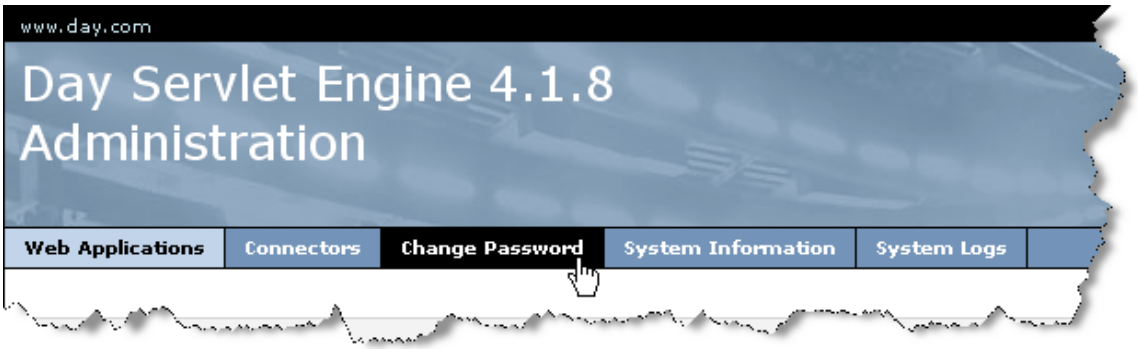
- e.g. http://localhost:4502/admin

2. Enter the default "User Name" (admin) and "Password" (admin) – then select OK.



CQSE login dialog

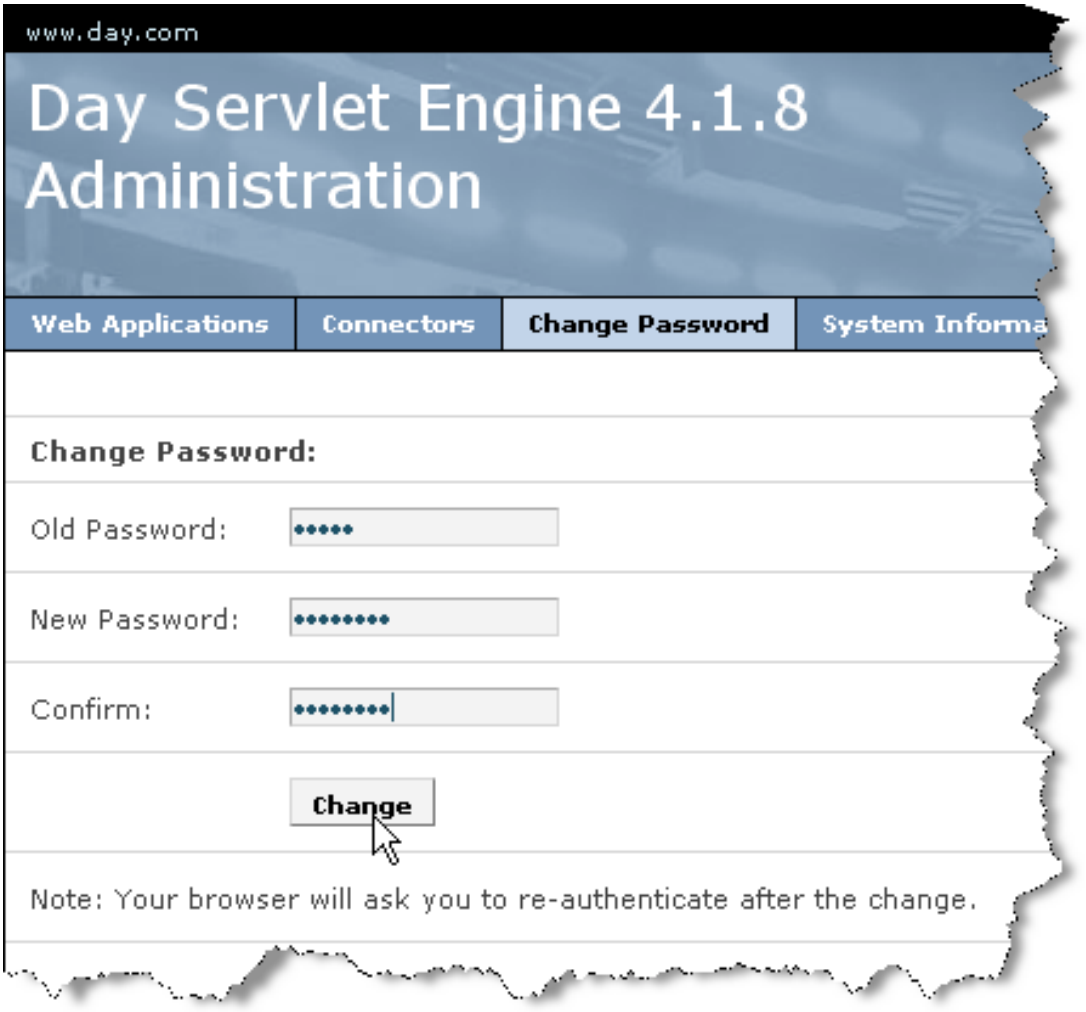
3. Select **Change Password**.



CQSE change password

Copyright 2010, Day Software AG, Switzerland

4. Enter the "Old Password" (admin), "New Password" (training), and "Confirm" (training) – then select **Change**.



CQSE change password confirm

Congratulations! You have successfully changed the CQSE default administrative password. Now focus on changing the content repository's (CRX) default administrative password.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

How to change the content repository's (CRX) default administrative password:

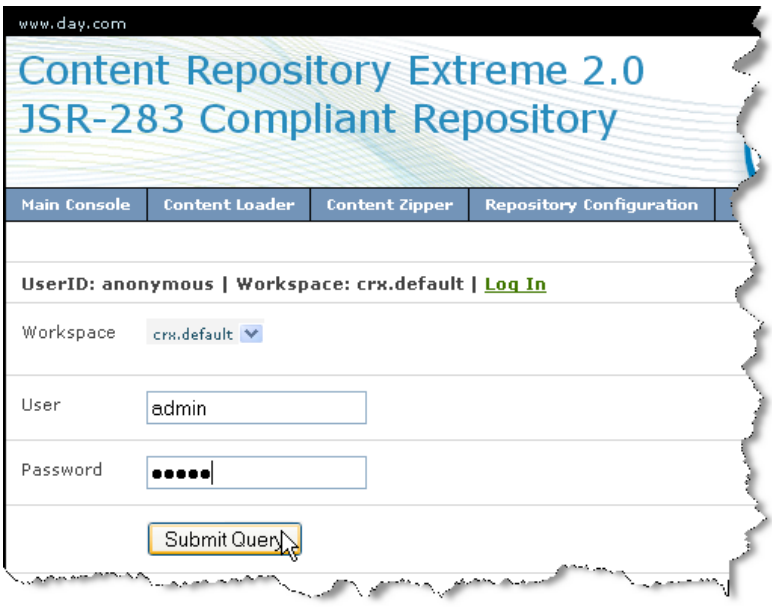
1. Navigate to the content repository (CRX) application.

   - e.g. http://localhost:4502/crx

2. Select Log In.

CRX login

3. Enter the "User" (admin) and "Password" (admin) – then select **Submit Query**.



CRX login dialog

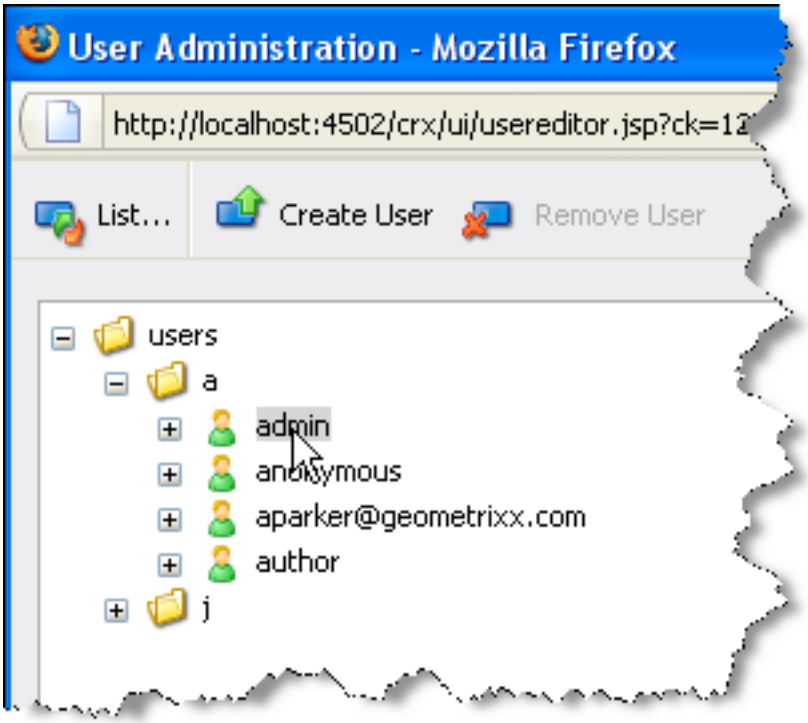
Copyright 2010, Day Software AG, Switzerland

4. Select User Administration.

CRX user administration

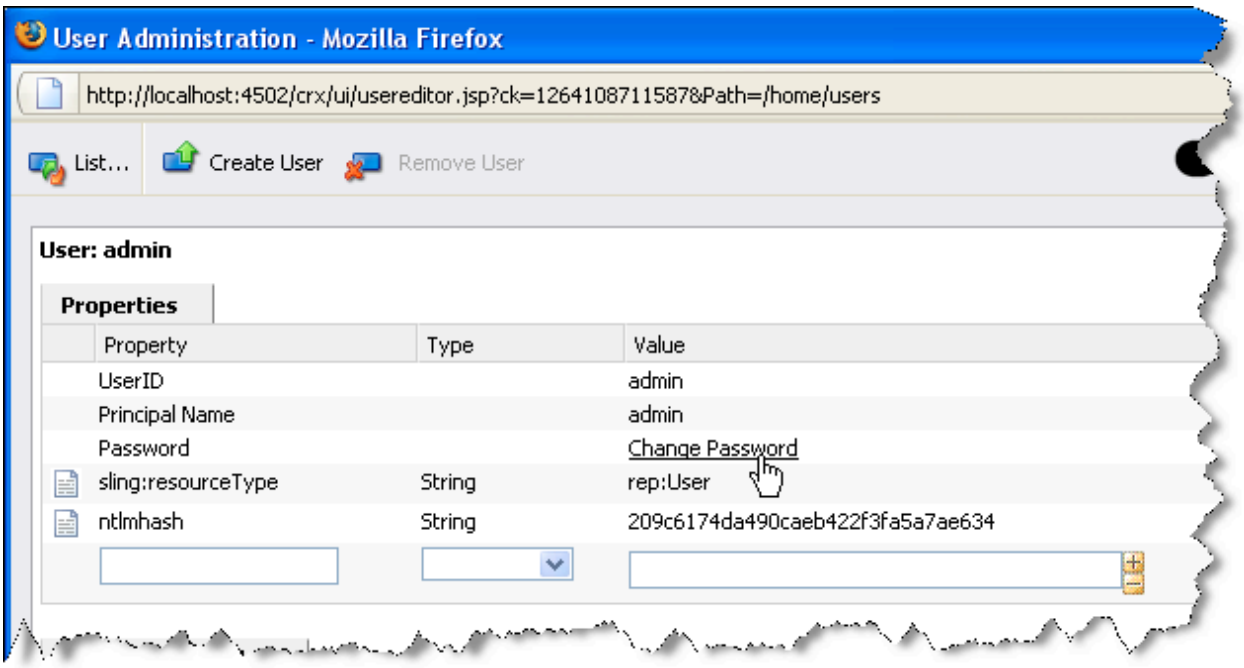
5. Navigate to and select the **admin user**.

Copyright 2010, Day Software AG, Switzerland



CRX admin user

6. Select **Change Password**.



CRX change password

Copyright 2010, Day Software AG, Switzerland

7. Enter the "new password" (training) and then retype where asked – then select **OK**.

CRX change password dialog

Congratulations! You have successfully changed the CRX default administrative password. Now focus on changing the Launchpad (Felix/Sling) application's default administrative password.

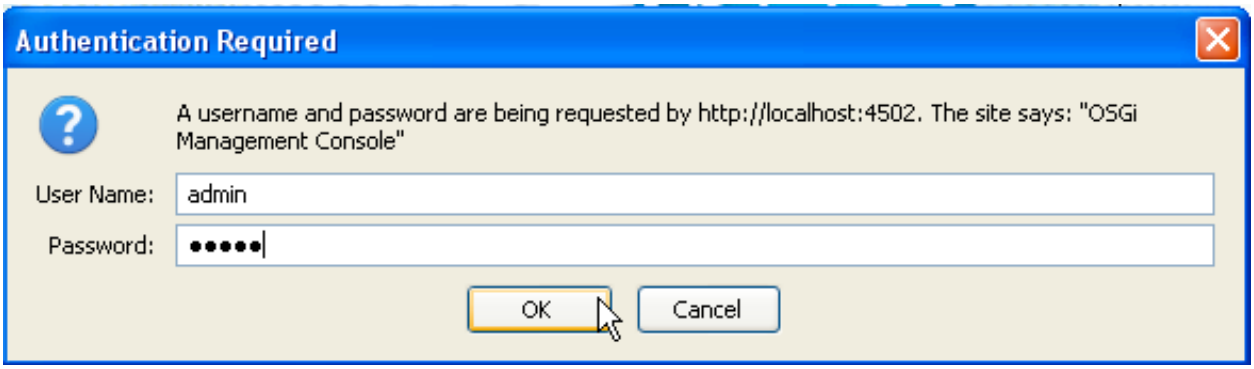
How to change the Launchpad (Felix/Sling) application's default administrative password:

1. Navigate to the Launchpad (Felix/Sling) application.

   - e.g. http://localhost:4502/system/console

2. Enter the default "User Name" (admin) and "Password" (admin) – then select **OK**.

Copyright 2010, Day Software AG, Switzerland



Launchpad login dialog

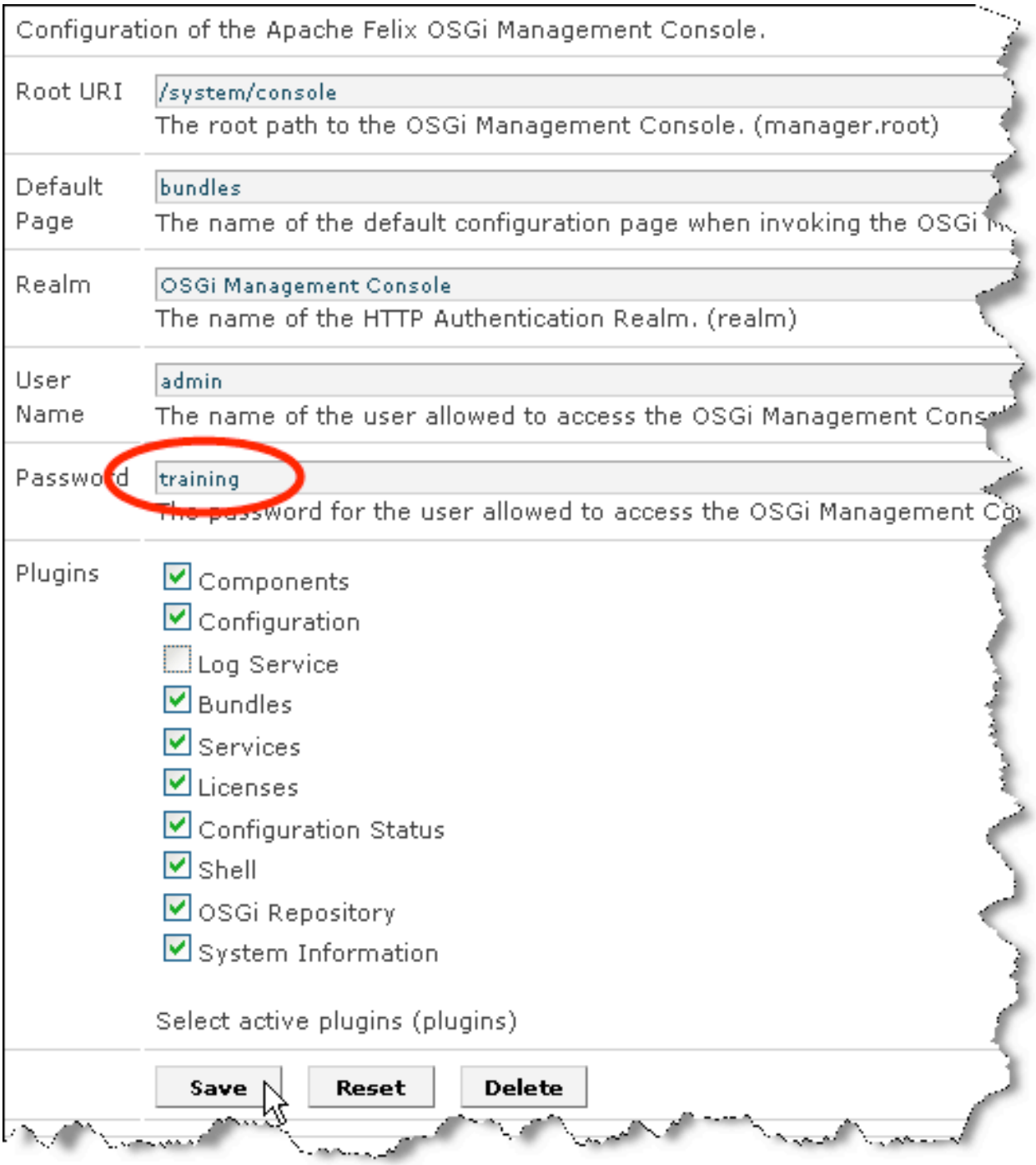
3. Select **Configuration**.

Launchpad configuration

4. Select **Apache Felix OSGi Management Console** from "Configuration" – then select Configure.

5. Enter the new "Password" (training) – then select **Save**.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com



Launchpad password dialog

Congratulations! You have successfully changed the Launchpad's default administrative password. Now focus on changing the Sling repository credential configuration.

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

How to change the Sling repository credential configuration:

1. Select CRX Sling Client Repository from "Configuration" in the Launchpad application – then select **Configure**.

2. Enter the new "Admin Password" (training) – then select **Save**.

Sling client repository admin password

Copyright 2010, Day Software AG, Switzerland

3. Validate changes have persisted properly by requesting the CQ application and login.

- e.g. http://localhost:4502/
- Username = admin
- Password = training

NOTE

It may take a minute or two for the changes to the CRX Sling Client Repository configuration to populate thoroughly.

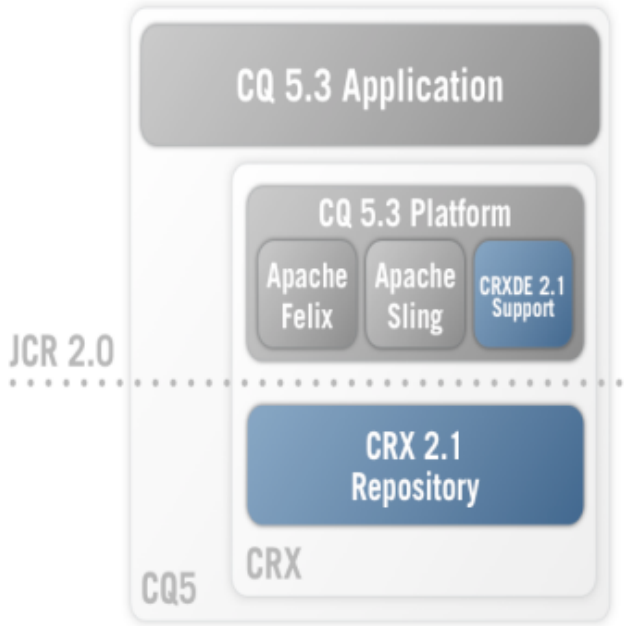
Congratulations! You have successfully changed the Sling repository credential configuration. Again, it is important for you to have a basic understanding of CQ security concerns. To learn more, you can always go to docs.day.com, and search for "cq security checklist".

Copyright 2010, Day Software AG, Switzerland

Appendix A
Exercise - Upgrading CQ 5.3 to a CRX 2.1 Repository

Goal

CQ 5.3 is bundled with CRX 2.0. The underlying CRX repository can be upgraded to version 2.1 for repository enhancements and/or bug fixes implemented in CRX 2.1, or enhanced support for CRX IDEs, CRXDE and CRXDE Lite:



The following instructions explain how upgrade your underlying CRX2.0 repository to CRX 2.1. To successfully complete and understand these instructions, you will need:

- A CQ5 quickstart JAR
- A valid CQ5 license key
- A JDK >= 1.5
- Approximately 800 MBs of free space
- Approximately 1 GB of RAM

Copyright 2010, Day Software AG, Switzerland

As with any upgrade, you should carefully consider value versus risk for your deployment. This includes testing the planned upgrade to ensure it passes your acceptance tests.

What will be Upgraded

The repository upgrade, as recommended here, has the following effect on the system. The following are ***upgraded***:

- **Infrastructure:** CRX Repository with all repository management and development tools
- **CQ5 Platform:** CRXDE support package for CRXDE Lite and CRXDE

The following are ***not upgraded:***

- Apache Sling and Apache Felix framework
- None of the CQ5 application components (bundles); with the exception of the CRXDE support package

The recommendation not to upgrade the Apache Sling and Felix frameworks, or any other application components, ensures that the stability of the CQ5 application as a whole is retained by minimizing the changes.

The following are ***removed:***

- CRXDE Lite was a separate web application in CQ 5.3 (CRX 2.0). It is now integrated into the main CRX web application.

Copyright 2010, Day Software AG, Switzerland

Upgrade procedure

For the following you will need an account with sufficient privileges for the various actions on files.

NOTE

The naming convention used here for the CRX *quickstart* file is:
crx-<version>-<edition>.jar.

For example, crx-2.1.0-enterprise.jar, where 2.1.0 would be the <version> and enterprise the name of the <edition>. The version might also include a date/timestamp. For this upgrade the version will be 2.1.0, be sure to substitute this and the name of your edition of CRX wherever appropriate.

To upgrade the CRX:

NOTE

In production, before any upgrade, take a full backup of your repository. Custom configurations made in server/runtime/0/_crx/WEB-INF/.. will be overwritten during the upgrade process. For example, changes to *web.xml* or *classes/indexing_config.xml.*

Save any changes you have made and reapply them after the upgrade.

12. Copy the CRX 2.1 quickstart jar file from ***USB>/distribution/cq53_crx2.1_upgrade*** to a suitable location on your hard disk. (e.g. C:\day\cq5\upgrade\crx).

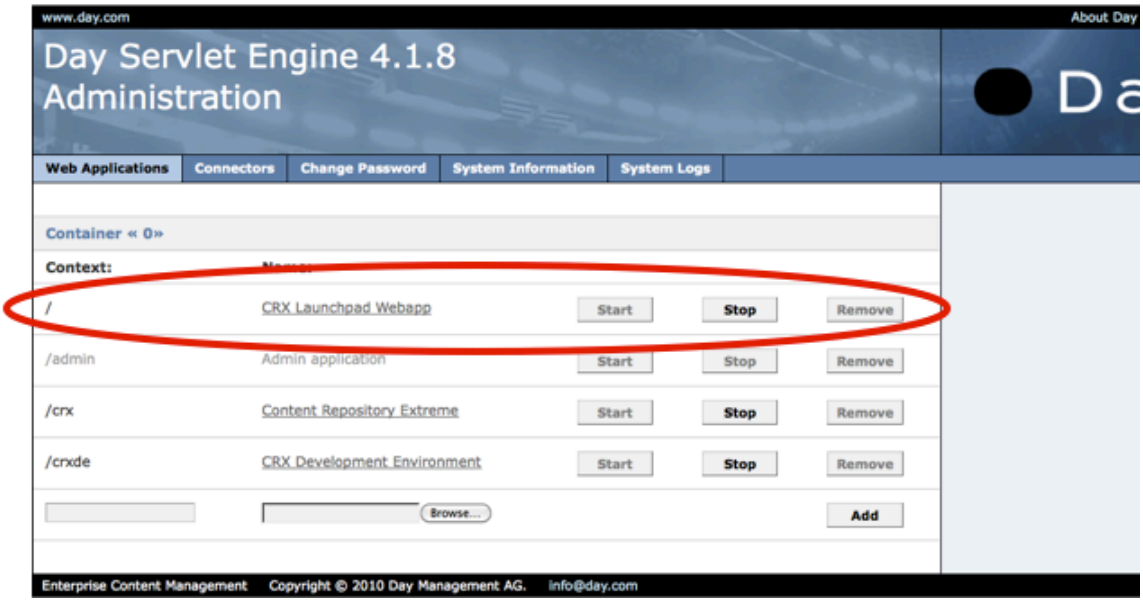
13. Unpack the CRX 2.1 jar file using the appropriate file name. e.g.,

    java -Xmx512m -jar crx-2.1.0.20100426-enterprise.jar -unpack

14. Open the CQSE admin console of your CQ 5.3 instance, for example:

    http://localhost:4502/admin

Copyright 2010, Day Software AG, Switzerland



15. Using the CQSE admin console, **Stop** the CRX Launchpad application

16. **Stop** and **Remove** both:

    - /crxde (the CRXDE application)
    - /crx (the CRX application)

17. Add a new:

    - /crx
      referencing the following file from the unpacked CRX 2.1:
      crx-quickstart/server/webapps/crx-explorer_crx.war

Copyright 2010, Day Software AG, Switzerland

Update the CRXDE server backend with the CQ 5.3 upgrade to CRX 2.1 package

NOTE

The CRXDE Support package is available on Day Package Share. Please consult the online documentation for instructions to download the package from the Day Package Share.

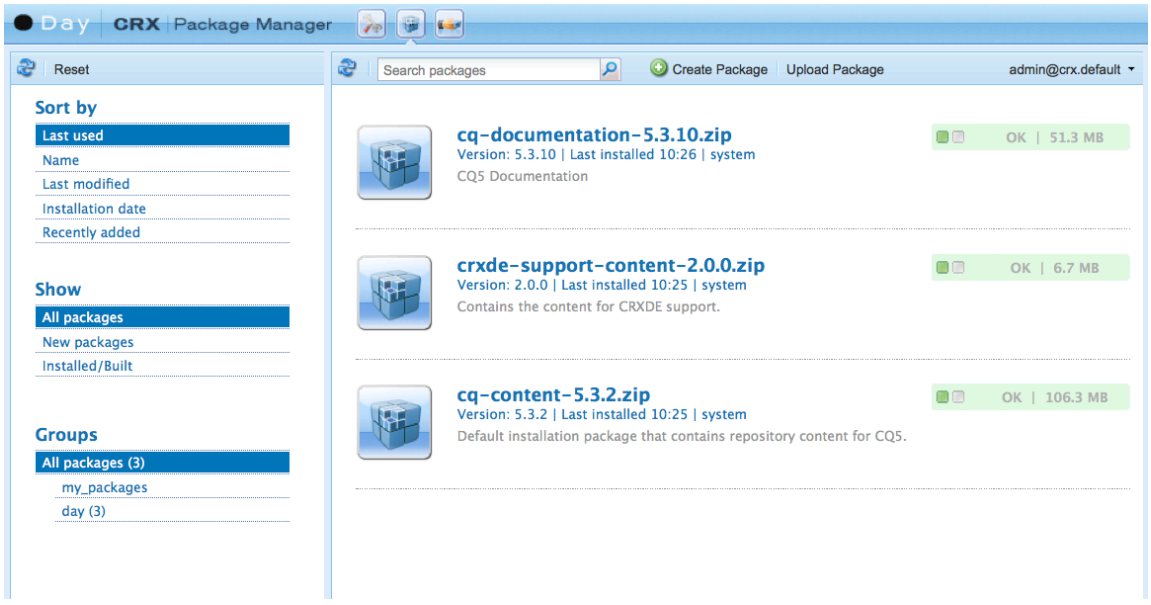
1. Copy the **cq53-update-crxdesupport-2.1.0.zip** file from the *<USB>/distribution/cq53_crx2.1_upgrade* to your hard drive. (e.g., C:\cq \upgrade).

2. Access the CRX Main Console and Log in using

   Username: admin

   Password: admin

3. Select the **Package Manager**



4. Upload and install the **cq53-update-crxdesupport-2.1.0.zip** package.

Copyright 2010, Day Software AG, Switzerland

5. Restart CQ to ensure that all OSGi bundles have been started.

NOTE

In case of problems with CQ startup, please open the Apache Felix Web Management Console (http://<host>:<port>/system/console) and check if all the bundles have been started. If a restart does not help, please start the bundles manually.

18. Confirm the upgrade of CRX by accessing:

- CRX
  for example, http://localhost:4502/crx/index.jsp
  The version details on the welcome screen will now show 2.1.
- CRXDE Lite
  for example, http://localhost:4502/crx/de/
  The version details on the welcome screen will now show 2.1.
- CQ
  use CQ to access your content, check everything is operating as expected.

CAUTION

You must test the operation of the upgraded instance; highly customized items may need to be upgraded separately.

NOTE

CRXDE Lite is now bundled with CRX (and not a separate webapp), access using /crx/de; for example, http://localhost:4502/crx/de/).

Copyright 2010, Day Software AG, Switzerland

Appendix B
Exercise - Clone an Author Instance to be a Publish Instance

Goal

The following instructions explain how to clone an Author instance. This is important because you will often need a Publish instance as part of your component testing. To successfully complete and understand these instructions, you will need:

- A stopped Author instance
- A valid CQ5 license key
- Approximately 800 MBs of free space

NOTE
The cloning method used here is a developer convenience. To clone a CQ5 instance for production, please consult the documentation and the CQ5 knowledge base.

What is an Publish instance?

A Publish instance is the CQ5 installation from which web site visitors will request pages.

How to clone an Author instance:

1. Make sure that your Author instance is stopped. You cannot successfully clone a running CQ5 instance.
2. Copy your author folder (e.g. C:/day/cq5/author).
3. Paste your author folder into the same folder structure (e.g. C:/day/cq5).
4. After the copy rename your author-copy folder to publish (e.g. C:/day/cq5/publish).
5. Rename the CQ5 quickstart JAR to *cq-publish-4503.jar*.

- cq = the application
- publish = the WCM mode it will run in (e.g. author or publish)

Copyright 2010, Day Software AG, Switzerland Rev 1.6 20101230 www.day.com

- 4503 = the port it will run in (e.g. any available port is acceptable)

NOTE
If no port number is provided in the file name, CQ5 will select the first available port from the following list: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, 7) 8085, 8) 8888, 9) 9362, 10) random.

6. Double-click the cq-publish-4503.jar file.

- A dialog will pop-up similar to the one below

CQ5 install/startup dialog

Congratulations! You have successfully cloned and started a CQ5 publish instance. To start CQ5 in the future, double-click the renamed CQ5 quickstart JAR file (e.g. cq-publish-4503.jar).

Copyright 2010, Day Software AG, Switzerland