eXo Content Reference Guide

eXo Content

Benjamin Paillereau (eXo Platform)

Chapter 1. WCM

Chapter 2. Misc

2.1. WCM FAQ

2.1.1. User FAQ

2.1.1.1. Is it possible to enable actions in 'Managed Sites' drive as we can do in a private drive?

Yes, of course you can. Please do follow these steps:

- Go to Sites Administration page, use the menu or this link http://localhost:8080/portal/private/acme/wcm/wcmAdmin
- Open Content Presentation -> Manage Drives
- Click on the edit icon of the drive you want to Manage Actions (Managed Sites in this case)
- Jump to the *Apply View* tab, check in *Admin* view
- Save
- Now go back to site explorer page. Go up if you are in a different drive (Green up arrow icon in the up-right corner).
- Go to Managed Sites drive and you will see the action, you can also switch to other views by choose them in the upper-left corner.

2.1.1.2. How to create an RSS Feeds in WCM 1.2?

If you have a freshly installed WCM, open http://localhost:8080/portal/private/acme/news Make sure that you created some documents in Acme driver (using the site explorer). It's an option on Parameterized Content List Viewer, and it's linked to the categories.

2.1.1.3. Where can I configure: "Vote for document" and "Comment this document" in WCM 1.2?

The Vote and Comment feature will be enabled in WCM 1.3

2.1.2. Technical FAQ

2.1.2.1. I get a strange error right at start-up.

You have to use Java 5. The use of Java 6 is not yet possible.

2.1.2.2. How do I configure the users who have access the Site Editor and Site Manager menus?

In "exo-workingxo-tomcatebappsortalEB-INFonfcmystem-configuration.xml", you find the

portal.creation.roles entry for the two menus "Sites Editor" and "Sites Administration" access permissions.

2.1.2.3. How do I configure the users who have access to the Mode Switch (Live Mode / Edit Mode)?

The access is given to a group. The default group is **:/platform/web-contributors**.

There are two different configuration files, that *both* give permission to access the button. The first configuration file only give access for a single site, the second for all sites.

-
  - Single site depending on the location of the file: "exo-tomcat/webapps/portal/WEB-INF/conf/portal/portal/portalname}/portal.xml" in "<edit-permission>**:/platform/web-contributors</edit-permission>" element. Only give access to the Mode Switch Button on the portal called portalname}.**
  - All sites: "exo-working/exo-tomcat/webapps/portal/WEB-INF/conf/wcm/system-configuration.xml", in **access.control.workspace.roles** parameter. Members of this groups have access on all sites.

2.1.2.4. Can I create a custom node type in a jar file ?

Yes, certainly you can. If you have read Kernel:Service Configuration for Beginners and Kernel:Service Configuration in Detail#Import you should know how to do it. In your jar file:

- Create a file /conf/portal/configuration.xml (all paths relative to your jar file root).
- Create other files necessary for a node type creation (below the /conf folder)
- You reference all files using the "jar:" namespace designator. Example: "jar:/conf/definitions/nodetype-mytype.xml"

(TODO: Article about Node types definition in DMS)

2.1.2.5. How can I create a site that is read-only?

1. The site can only be changed by the user who has the permissions to do so. Anonymous users have in general no permission and cannot change anything (any content or navigation) on your site. You may also create users without any permissions to change the site, which is very often the case in production environments.

1. If the question refers to "does not write anything to the JCR", we have to say that is impossible. Everything is saved in the repository (users, preferences, navigation, content). If you have a site which only shows content it's probable there isn't written much to the repository. But it's probable that the future statistics feature of WCM writes in the JCR. Furthermore you might have portlets that write in the JCR, for example if the portlet use forms and stores the content in an JCR node.

2.1.2.6. How to avoid the minimum width of 300px of a portlet?

The HTML generated by this portlet is the following:

```
<div class="UISingleContentViewerPortlet" style="min-width: 300px;" id="UISingleContentViewerPortlet">
```

You see it's hard-coded in the html but you can use the key word *important*. Add the following line in your css file: *div.UISingleContentViewerPortlet { min-width: 0px !important; }*

2.1.2.7. How can I make the publication workflow user dependent?

Since WCM 1.2 you can define user-dependent publication workflows. Normally you see a *Stage and Version publication* lifecycle but this can be changed We will provide a management tools to manage publication workflows, and you can use it to control your publication lifecycles. But it isn't yet available in WCM1.2.

2.1.2.8. Where is all the default content of the WCM main page stored?

We use Single Content Viewer portlets to present these contents. You can see the contents of this portlets using the Site Explorer, Driver Managed Sites. For example in order to see the content of the logo open *classic/web contents/site artifacts/Logo* (in WCM1.2).

2.2. Friendly URLs

2.2.1. Purpose

With eXo WCM you can simply use the Content List Viewer portlet (CLV) in conjunction with the Parameterized Content Viewer portlet (PCV) in order to simply display a list of content and their details. By default, WCM content URLs are pointing directly to the file located within the JCR storage structure, so the URL path contains technical elements inside.

For example, the URL of a WCM content can be as complex as that:

```
http://localhost:8080/portal/private/vitrines/products/presentation/pcv/repository/collaboration/sites%20content
```

In this article, we are going to see how to remove all technical information from URLs and get clean, friendly and REST-like bookmarkable URLs.

Using the tips described in this article, you can get URLs that are as clean as the following:

```
http://localhost:8080/portal/article/My-Personal-Article
```

With a little extra effort, you can get this:

```
http://localhost:8080/portal/article/2009/11/27/My-Personal-Article
```

2.2.2. URL Rewrite Filter

Download and unzip the http://tuckey.org/urlrewrite/#download zip file and put the jar file somewhere in the server classpath.

Add the following lines to the exoplatform portal.war web.xml file.

```
<filter>
        <filter-name>UrlRewriteFilter</filter-name>
```

```
          <filter-class>org.tuckey.web.filters.urlrewrite.UrlRewriteFilter</filter-class>
       </filter>

...
 <filter-mapping>
          <filter-name>UrlRewriteFilter</filter-name>
          <url-pattern>/*</url-pattern>
  </filter-mapping>
```

Make sure to put the filter mapping **after** the other filters otherwise it will not work as expected.

UrlRewriteFilter searches the configuration file "urlrewrite.xml" in the WEB-INF directory. Use regular or wildcard expressions for the parameters.

2.2.2.1. Rewriting Rule Configuration File

Put a urlrewrite.xml file in the WEB-INF directory of the portal.war. Define the following rewriting rule in the file:

```
<rule>
    <note>
        The rule means that requests to /article/myarticle will be forwarded to
        /public/classic/mypage/repository/collaboration/myfolder/myarticle
        the url will be rewritten.
    </note>
    <from>/article/(.*)</from>
    <to>/public/classic/mypage/repository/collaboration/myfolder/$1</to>
</rule>
```

Make sure that the target URL points to the right portal site, page, repository workspace and JCR folder paths where your contents are located.

2.2.3. Demonstration

Now let's do a simple page to show how all this works.

2.2.3.1. Page Layout

Create a new page named 'mypage' in WCM, at the root of the portal navigation and put a CLV and a PCV inside.

2.2.3.2. Configuration

Configure the CLV to point to the right JCR path (where the is content) and the target page to 'mypage' because the CLV and PCV are both on the same page.

2.2.3.3. CLV Template Modification

Open the Sites administration / DMS Administration drive. Then go to the "/exo:ecm/views/templates/Content List Viewer/list-by-folder" folder. Edit the UIContentListPresentationDefault.gtmpl content, which is the template we choose for the CLV (see also WCM Templates).

Locate the following block of code and insert the line below:

itemLink = "/portal/article/$itemName"

```
...
for (Node node : currentPageData) {
   def viewNode = Utils.getNodeView(node);
   if (viewNode == null) continue;
   String itemLink = uicomponent.getURL(viewNode);
   String itemName = viewNode.getName();
   itemLink = "/portal/article/$itemName"
   String itemSummary = uicomponent.getSummary(viewNode);
...
```

Save and restart the server (to clear the template cache).

2.2.4. Conclusion

That's it! Now you have clean and friendly URLs for your contents. Look to URL in the address bar and to the status bar when moving the mouse over content titles in the CLV. You can see that URLs are short and clean.

2.2.5. Overcome the Restrictions

Some restrictions apply to this URL rewriting trick, mainly the highly generic URL that cover all kind of use-cases is replaced by a more restricted single use-case URL. This solution entails several drawbacks:

- The page name, the workspace and the folder are hard-coded in the URL rewriting rule. This rewriting rule is applied to all CLV instances that use the modified viewer template. In fact in the viewer template "UIContentListPresentationDefault.gtmpl" you hard-code the shortcut URL "/portal/article/$itemName" for which the rewriting rule is applied.
  - All articles pointed by any of these CLV instances must be in the same workspace (in this example "collaboration").
  - All articles pointed by any of these CLV instances must be in the same folder (in this example "myfolder").
  - All articles pointed by any of these CLV instances must be in the same portal/site (in this example "classic").
  - By shortcutting the URL you can only point to the same page. Pointing to other pages from a CLV that uses this template will not work.

Finally the hint after listing the restrictions: Better define a specific viewer template for this trick in order not to destroy the generic URLs of the default CLV template. Even better create several new templates as you might wish to point to different folders or pages. In this case you need several URL rewriting rules.

2.3. How to create new publication in WCM

Note

Since WCM 1.2.2

2.3.1. Create new Java project

- Create new Java project with the structured like this:

- **Note**: src/main/java and src/main/resources are source folders

2.3.2. Configuration

- In configuration.xml, there is two required mandatory configurations:

For the publication plugin

```
<external-component-plugins>
 <target-component>org.exoplatform.services.wcm.publication.WCMPublicationService</target-component>
 <component-plugin>
  <name>Date time publication</name>
  <set-method>addPublicationPlugin</set-method>
  <type>org.exoplatform.services.wcm.publication.lifecycle.datetime.DateTimePublicationPlugin</type>
  <description>This publication lifecycle publish a web content or DMS document to a portal page with more state
  <init-params>
   <values-param>
    <name>repository</name>
    <value>repository</value>
   </values-param>
   <values-param>
    <name>workspace</name>
    <value>collaboration</value>
   </values-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

- **Note**:
  - target-component: always is org.exoplatform.services.wcm.publication.WCMPublicationService.
  - name: your publication's name.
  - description: your publication's description.
  - set-method: always is addPublicationPlugin.
  - type: your publication's class.
  - init-params: not required, you can use it for your publication plugin or not.

And for the publication nodetype

```
<external-component-plugins>
 <target-component>org.exoplatform.services.jcr.RepositoryService</target-component>
 <component-plugin>
  <name>add.nodeType</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.services.jcr.impl.AddNodeTypePlugin</type>
  <priority>99</priority>
  <init-params>
   <values-param>
    <name>autoCreatedInNewRepository</name>
```

```
    <description>Node types configuration file</description>
    <value>jar:/conf/wcm-datetime-publication-nodetypes.xml</value>
   </values-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

- **Note**: Just change the name of the xml file to match with your nodetype's configuration file.

2.3.3. Extends or implement the publication plugin

- If you want to re-use the existing publication in WCM, you can extend one of these plugins:
  - org.exoplatform.services.wcm.publication.lifecycle.stageversion.StageAndVersionPublicationPlugin
  - org.exoplatform.services.wcm.publication.lifecycle.simple.SimplePublicationPlugin
- If you want to create a new publication, you have to extend **org.exoplatform.services.wcm.publicationWebpagePublicationPlugin**, and you have to overwrite all abstract methods. Important method:
  - **getLifecycleType()**: return the nodetype of the publication

2.3.4. Create new User interface (UI) for new publication

- All UI classes of the new publication are put in src/main/java, and all groovy templates are put in src/main/resources. If you want to use the template, use "classpath:path/in/src/main/resource/to/the/templates" (without quote) in ComponentConfig.

2.4. How to create a site template in WCM

2.4.1. Add a new template for portal

- Open this file <TOMCAT-HOME>/portal/WEB-INF/conf/uiconf/portal/webui/portal/PortalTemplateConfigOption.groovy

Normally, it's look like this

```
import java.util.List;
import java.util.ArrayList;
import org.exoplatform.portal.webui.portal.PortalTemplateConfigOption ;
import org.exoplatform.webui.core.model.SelectItemCategory;
List options = new ArrayList();
SelectItemCategory acme = new SelectItemCategory("ACMESite");
acme.addSelectItemOption(new PortalTemplateConfigOption("ACME Site", "acme", "ACME Site", "ACMESite").addGroup("
options.add(acme);
return options ;
```

And you have to add some lines like this before return.

```
SelectItemCategory test = new SelectItemCategory("TestTemplate");
test.addSelectItemOption(new PortalTemplateConfigOption("Test Template", "test", "Test Template", "TestTemplate"
options.add(test);
```

2.4.2. Add resource bundles for new template

- Open <TOMCAT-HOME>/portal/WEB-INF/classes/locale/wcm/webui*en.xml (and other webui.xml files also), and add these*

```
<TestTemplate>
 <label>Test template</label>
</TestTemplate>
```

1 Update illustration image for new template

- Open <TOMCAT-HOME>/eXoWCMResources/skin/DefaultSkin/wcm-portal/Stylesheet.css to update the stylesheet

```
.UIItemSelector .ItemDetailList .TemplateContainer .TestTemplateImage {
 background:transparent url(background/testtemplate.png) no-repeat center;
 height: 235px;
 width: 296px;
 margin: 3px auto;
}
```

- **Note**:
  - The class name is the name of new template appends with "Image"
  - Please don't forget to add new image into <TOMCAT-HOME>/eXoWCMResources/skin/DefaultSkin/wcm-portal/background/ folder

1 Update pages and portlet preferences

- Put some xml files to <TOMCAT-HOME>/portal/WEB-INF/conf/portal/portal/template/test/ folders
  - navigation.xml
  - pages.xml
  - portal.xml
  - portlet-preferences.xml
- **Note**: owner id in these files always is @owner@
- Hint: you can make a copy from acme template files and edit anything you want

1 Update pre-defined contents data (like logo, footer, navigations, ...)

- To create your own contents, you can do as follow:
  - Run WCM
  - Go to Site Explorer and create new Web contents
  - Publish it by Manage publication feature
  - Export it (with version history), and you will have some xml files
- Put them in portal/WEB-INF/conf/wcm/artifacts/site-resources/test-templates/ folder

1 Update pre-defined contents configuration

- Make a copy of file <TOMCAT-HOME>/portal/WEB-INF/conf/wcm/deployment/template-deployment-configuration.xml, and modify it

```
<external-component-plugins>
 <target-component>org.exoplatform.services.wcm.portal.artifacts.CreatePortalArtifactsService</target-component>
 <component-plugin>
  <name>Initial webcontent artifact for each site</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.services.wcm.webcontent.InitialWebContentPlugin</type>
  <description>This plugin deploy some initial webcontent as site artifact to site artifact folder of portal whe
  <init-params>
   <object-param>
    <name>Portal logo data</name>
    <description>Deployment Descriptor</description>
    <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
     <field  name="target">
      <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
       <field  name="repository"><string>repository</string></field>
       <field  name="workspace"><string>collaboration</string></field>
       <field  name="nodePath"><string>/sites content/live/{portalName}/web contents/site artifacts</string></fi
      </object>
     </field>
     <field  name="sourcePath"><string>war:/conf/wcm/artifacts/site-resources/test-templates/Logo.xml</string></
    </object>
   </object-param>
   <object-param>
    <name>Portal signin data</name>
    <description>Deployment Descriptor</description>
    <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
     <field  name="target">
      <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
       <field  name="repository"><string>repository</string></field>
       <field  name="workspace"><string>collaboration</string></field>
       <field  name="nodePath"><string>/sites content/live/{portalName}/web contents/site artifacts</string></fi
      </object>
     </field>
     <field  name="sourcePath"><string>war:/conf/wcm/artifacts/site-resources/test-templates/Signin.xml</string>
    </object>
   </object-param>
   ...
  </init-params>
 </component-plugin>
</external-component-plugins>
```

- **Note**: portalName} will be replaces with the name of portal when a portal is created

2.5. How to create and deploy pre-defined contents in WCM

2.5.1. Create contents

- Open <TOMCAT-HOME>/portal/WEB-INF/conf/configuration.xml, and replace this one

```
<import>war:/conf/wcm/publication-configuration.xml</import>
```

by this one

```
//<import>war:/conf/wcm/publication-configuration.xml</import>
```

- Run WCM with a clean database
- Go to Site explorer and choose any drive but we recommend you should go to Sites management drive.
- Click on Add content and choose Free layout web content as template.
- Input your HTML, CSS and Javascript data then Save
- Switch to System tab, click on Export node
- Click Export to export content, and you will have an XML file.
- Do these steps again and again to export all contents you want.
- Open <TOMCAT-HOME>/portal/WEB-INF/conf/configuration.xml, and replace this one

```
//<import>war:/conf/wcm/publication-configuration.xml</import>
```

by this one

```
<import>war:/conf/wcm/publication-configuration.xml</import>
```

2.5.2. Deploy contents

- You can setup for the deployment like this

```
<external-component-plugins>
 <target-component>org.exoplatform.services.wcm.deployment.WCMContentInitializerService</target-component>
 <component-plugin>
  <name>Content Initializer Service</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.services.wcm.deployment.plugins.XMLDeploymentPlugin</type>
  <description>XML Deployment Plugin</description>
  <init-params>
   <object-param>
    <name>ACME Logo data</name>
    <description>Deployment Descriptor</description>
    <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
     <field name="target">
      <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
       <field name="repository"><string>repository</string></field>
       <field name="workspace"><string>collaboration</string></field>
       <field name="nodePath"><string>/sites content/live/acme/web contents/site artifacts</string></field>
```

```
      </object>
     </field>
     <field  name="sourcePath"><string>war:/conf/wcm/artifacts/site-resources/acme/Logo.xml</string></field>
    </object>
   </object-param>
   <object-param>
    <name>ACME Signin data</name>
    <description>Deployment Descriptor</description>
    <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
     <field name="target">
      <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
       <field name="repository"><string>repository</string></field>
       <field name="workspace"><string>collaboration</string></field>
       <field name="nodePath"><string>/sites content/live/acme/web contents/site artifacts</string></field>
      </object>
     </field>
     <field  name="sourcePath"><string>war:/conf/wcm/artifacts/site-resources/acme/Signin.xml</string></field>
    </object>
   </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

- **Node**:
  - target-component: always is org.exoplatform.services.wcm.deployment.WCMContentInitializerService
  - component-plugin/name: anything
  - component-plugin/set-method: always is addPlugin
  - component-plugin/type: always is org.exoplatform.services.wcm.deployment.plugins.XMLDeploymentPlugin
  - component-plugin/description: anything
  - object-param/name: the name of the content will be deployed (unique)
  - object-param/description: anything
  - object/field repository: the current repository name
  - object/field workspace: the current workspace name
  - object/field nodePath: the folder which content will be deployed inside
  - object-param/field sourcePath: the path to the XML file
- **Hint**: You can see an example in this file <TOMCAT-HOME>/portal/WEB-INF/conf/wcm/deployment/acme-deployment-configuration.xml

2.6. How to use FCKEditor in WCM

2.6.1. FCKEditor's plugins in WCM

In WCM, we developed some new plugins to communicate media files between client and server. If you need something else, please tell us and we can discuss about that :)

2.6.1.1. Insert images

> **Note**
>
> *Deprecate since WCM 1.2*

2.6.1.2. Insert documents

> **Note**
>
> *Deprecate since WCM 1.2*

2.6.1.3. Insert portal links

2.6.1.4. Insert gadgets

2.6.1.5. Insert contents

> **Note**
>
> *Since WCM 1.2*

2.6.2. FCKEditor's toolbar in WCM

We also provide some default toolbars for you. Of course you can customize it as you want.

2.6.2.1. FCKEditor plugins's icon

Insert images

 Insert documents

 Insert portal links

 Insert gadgets

 Insert contents

2.6.2.2. CompleteWCM

exoconfig.js

```
FCKConfig.ToolbarSets["CompleteWCM"] = [
        ['Source','Templates','-','FitWindow','ShowBlocks'],
        ['Cut','Copy','PasteText','-','SpellCheck','-','Undo','Redo'],
        ['WCMInsertGadget','Flash','Table','SpecialChar', 'WCMInsertContent'],
        '/',
        ['Bold','Italic','Underline','StrikeThrough','-','JustifyLeft','JustifyCenter',
         'JustifyRight','JustifyFull','-','OrderedList','UnorderedList','-','TextColor','BGColor','-','RemoveFor
```

```
        ['Link','WCMInsertPortalLink','Unlink','Anchor'],
        '/',
        ['Style','FontFormat','FontName','FontSize']
] ;
```

> **Note**
>
> *Before WCM 1.2*

> **Note**
>
> *Since WCM 1.2*

2.6.2.3. BasicWCM

exoconfig.js

```
FCKConfig.ToolbarSets["BasicWCM"] = [
        ['Source','-','Bold','Italic','Underline','StrikeThrough','-','OrderedList','UnorderedList','Outdent','I
        ['JustifyLeft','JustifyCenter','JustifyRight','JustifyFull'],
        ['Blockquote','-','Link','Unlink','WCMInsertPortalLink','WCMInsertContent','-','FitWindow','ShowBlocks']
        ['Style','FontFormat','FontName','FontSize']
] ;
```

> **Note**
>
> *Before WCM 1.2*

> **Note**
>
> *Since WCM 1.2*

2.6.2.4. SuperBasicWCM

exoconfig.js

```
FCKConfig.ToolbarSets["SuperBasicWCM"] = [
     ['Source','-','Bold','Italic','Underline'],
     ['-','JustifyLeft','JustifyCenter','JustifyRight','JustifyFull'],
     ['-','Link','Unlink','WCMInsertPortalLink','WCMInsertGadget','WCMInsertContent'],
] ;
```

> **Note**

Before WCM 1.2

Note

Since WCM 1.2

2.6.3. FCKEditor and dialogs

If you want to use FCKEditor in WCM's dialogs, you can do as follow:

```
String htmlContent = "";
String[] htmlArguments = ["jcrPath=/node/node_property", "options=toolbar:CompleteWCM", htmlContent];
uicomponent.addWYSIWYGField(htmlWYSIWYGFormId, htmlArguments) ;
```

You can change somethings as you want:

- htmlContent: the pre-defined values for FCKEditor
- node*property: the property where the content is stored. You also can save the content in the nt:file, but you have add some new lines like this **before** the code above*

```
String[] htmlHiddenField1 = ["jcrPath=/node/file.txt", "nodetype=nt:file", "defaultValues=default.html"];
String[] htmlHiddenField2 = ["jcrPath=/node/file.txt/jcr:content", "nodetype=nt:resource", "visible=false"];
String[] htmlHiddenField3 = ["jcrPath=/node/file.txt//jcr:content/jcr:encoding", "visible=false", "UTF-8"];
String[] htmlHiddenField4 = ["jcrPath=/node/file.txt//jcr:content/jcr:lastModified", "visible=false"];
String[] htmlHiddenField5 = ["jcrPath=/node/file.txt//jcr:content/dc:date", "visible=false"];
String[] htmlHiddenField6 = ["jcrPath=/node/file.txt//jcr:content/jcr:mimeType", "visible=false", "text/plain"];
uicomponent.addHiddenField("htmlHiddenField1", htmlHiddenField1);
uicomponent.addHiddenField("htmlHiddenField2", htmlHiddenField2);
uicomponent.addHiddenField("htmlHiddenField3", htmlHiddenField3);
uicomponent.addCalendarField("htmlHiddenField4", htmlHiddenField4);
uicomponent.addCalendarField("htmlHiddenField5", htmlHiddenField5);
uicomponent.addHiddenField("htmlHiddenField6", htmlHiddenField6);
```

- CompleteWCM: You can change and use other FCKEditor's toolbar of WCM like **BasicWCM**, **SuperBasicWCM**

Hint: You can learn more about the dialog here ECM Template Management

Chapter 3. WebDAV

3.1. Using WebDAV

Accessing into a site by WebDAV view, you also normally see all default folders of a site. Manipulating on WebDAV likes manipulating on folders means you also can copy/paste, list, rename folders...as system directories.

In here, users can take all actions with WebDAV as Window such as: copy, paste, delete files and documents:

- Users can copy a HTML file and then paste in the **Web Content** Folder to create a new web content.
- Users can copy a CSS file on **CSS** folder to change stylesheet for a site.
- Users can copy a JS file in **JS** folder.
- Users can copy some documents and then paste on the **Documents** folder.

Reference

1. We can see how to use WebDAV at WCM:WebDAV in WCM

3.2. Using WebDAV with Frontpage

Install FrontPage 2003.

2. Open Microsoft FrontPage 2003.

3. On the File menu, select 'Open Site'. The Open Site window will appear:

4. Enter ***your site's WebDav address*** into the **Site Name** text box and click **Open**. The **Web Site Setup** window appears as:

- Select the 'WebDAV' option and the 'Encrypted Connection Required' (SSL) checkbox, then click the **Next** button.

The Import Web Site Wizard window will appear:

- Select a local folder where you'd like to store the web site on your computer. This is where you'll be able to edit files before uploading back to your web site. We recommend you store the site in a new folder within your **My Web Sites** folder:

1 Click the **Browse** button

2. Select My Documents then open the **My Web Sites** folder.

3. Click the **New Folder** button, enter the name of your site and click **OK**.

4. Click the **Open** button.

5. Click the **Next** button.

6. Click **Finish**.

1. split screen appears showing the local site and the remote web site (your site on the web server) as picture:

- On the right column of a screen, it will display all folders of your website on server. We will choose all folders and then click to move all folders on server to location.

- Then you can choose web content, CSS, and JS to edit this location.

***Users can edit web content by editor FrontPage:*_** *5. If you want to edit a content, do as follows :*

- Click the **Web Content** folder on the left column.
- Click the **site artifacts** folder -> Click the **homepage** folder-> Click the **Default** folder--> Double click the **Default.html** file. It will display a form to edit HTML file. In here we can see, I have just edited the "ProductSlogan" class in the 'Default.html' file:

6. We can edit the content of CSS as below:

- Choose the **WebSite** tab-> click **CSS folder** -> Double click the **Default.css** file. It will display a form to edit CSS file. We can edit this file (In here i have just changed background of homepage from White color to Blue color and height of this banner from 258px to 500px) and then click the

  icon to save all changes. We have a view with this picture:

7. Choose the HTML **tab-> then click** $_{\textbf{Format}}$ **on the menu -> choose** $_{\textbf{Style Sheet Links}}$**-> Click** $_{\textbf{Add}}$ **and then choose the path-> choose edited CSS file to connect with this HTML file.**

We can see by the following interface:

+Note:_+* After saving in HTML file, you should remove all codes generated when you make style sheet link.

8. After editing all files which you want to, click the icon to synchronize all changes on location to server.

9. After synchronizing, we can browse the website on explorer to see all changes as picture below:

- ***Users can create a web content by editor FrontPage:*_***

From split interface, on the "local files" interface --> Click the icon, it will display an interface to create a HTML file as below:

Create HTML code on this and then click **File** --> **Save as** and it will display a form to choose which folder you want to save on local site as below:

Finally, you can come back WCM and refresh this on explorer, you can see this content on **sites explorer** and this content have structures as all contents in sites explorer as below:

3.3. Using WebDAV with Dreamweaver

Note

This tutorial is only for 1.0.x and later version

1 Install Dreamweaver CS3.

2. Open Dreamweaver CS3.

3. On Menu bar choose **Site**--> **New Site** then it will display a form to create a new Dreamweaver site definition:

In here you can take some actions to connect between server site and local site to edit the web content, CSS file and JS file with Dreamweaver editor.

4. On the **Site name** textbox , here is the name of site which you want to do with. In here I type "WCM*eXo*site"

5. On **Local root folder** , here you click on the icon to choose which folder on local computer that you want to store and manage site.

6. On **HTTP address** textbox you type the public address of your website, in here I type "http://localhost:8080/portal/public/classic".

7. Then choose the **Remote info** in **Advanced** tab. In the **Access** textbox choose **WebDAV**.

8. In URL textbox type WebDAV link of your site.

9. In the **Login** and **Password** textbox, type the username and password which you use to login on your site.

10. After that, click on the **Test** button to test the connection with server. If it is connected, it will display:

11 After being connected, you should see a list of files in the **Remote Site** panel:

12. Select the top-most folder in either the **Remote Site** or **Local Files** panel.

13. Click the down arrow button to "Get" files from the **Remote Site**. You will see an alert that **"you are copying the entire site"**. Click the **OK** button.

14. You can click **No** when prompted about dependent files when getting the whole site.

15. You should now have list of files in your Local Folder:

You now have all your site files and are one step away from editing your pages. If you open any page file, it doesn't look like it does on the browser. Set up your local Templates folder next. * **We can edit pages with Dreamweaver**

If you want to edit a content, do as follows:

- Click the **Web Content** folder on the local files panel.
- Click the **site artifacts** folder -> Click the **banner** folder-> Click the **Default** folder--> Double click the **Default.html** file. It will display a form to edit HTML file. In here we can see, i have just edited the class "BannerTitle" in the 'Default.html' file:

After finishing the edition of HTML file, you choose **File**--> choose **Save** to save all changes in this file. After that you can see all changes on Browser as:

- **We can edit CSS file with Dreamweaver**

If you want to edit CSS file, do as below:

- Click **Web Content** folder on the local files panel.
- Click the **site artifacts** folder -> Click the **banner** folder -> Click the **CSS** folder --> Double click the **Default.css** file. It will display a form to edit HTML file. In here we can see, I have just edited the class "BannerTitle" in 'Default.css' file and from **color: #bebebe;** to **color: red;** and **font-family: Tahoma;** to **font-family: Arial;** as picture below:

- After finishing all changes-> click **File** -> click **Save** to save all changes in CSS file. And then we can come back HTML file to see all changes in CSS file which will affect to HTML.

As picture above, we can see that the content was changed as all changes in CSS file.

- **We can create web content with Dreamweaver**

If you want to create web content, do as below:

- Click the **Web Content** folder on the local files panel.
- Click **File**-> Click **New**-> it will display a form to create the HTML code.
- Create HTML code -> then click **File** -> Click **Save**.
- Then click the

  synchronize icon to put this content to server file, as you can see here:

After that, we can come back to the browser to run eXo WCM, in which we can see this content on **Sites explorer** as:

In here I have just created new web content named *New*Content. *After saving and synchronizing the server file,*

*New*Content will become web content with same structures with other contents in sites explorer as:

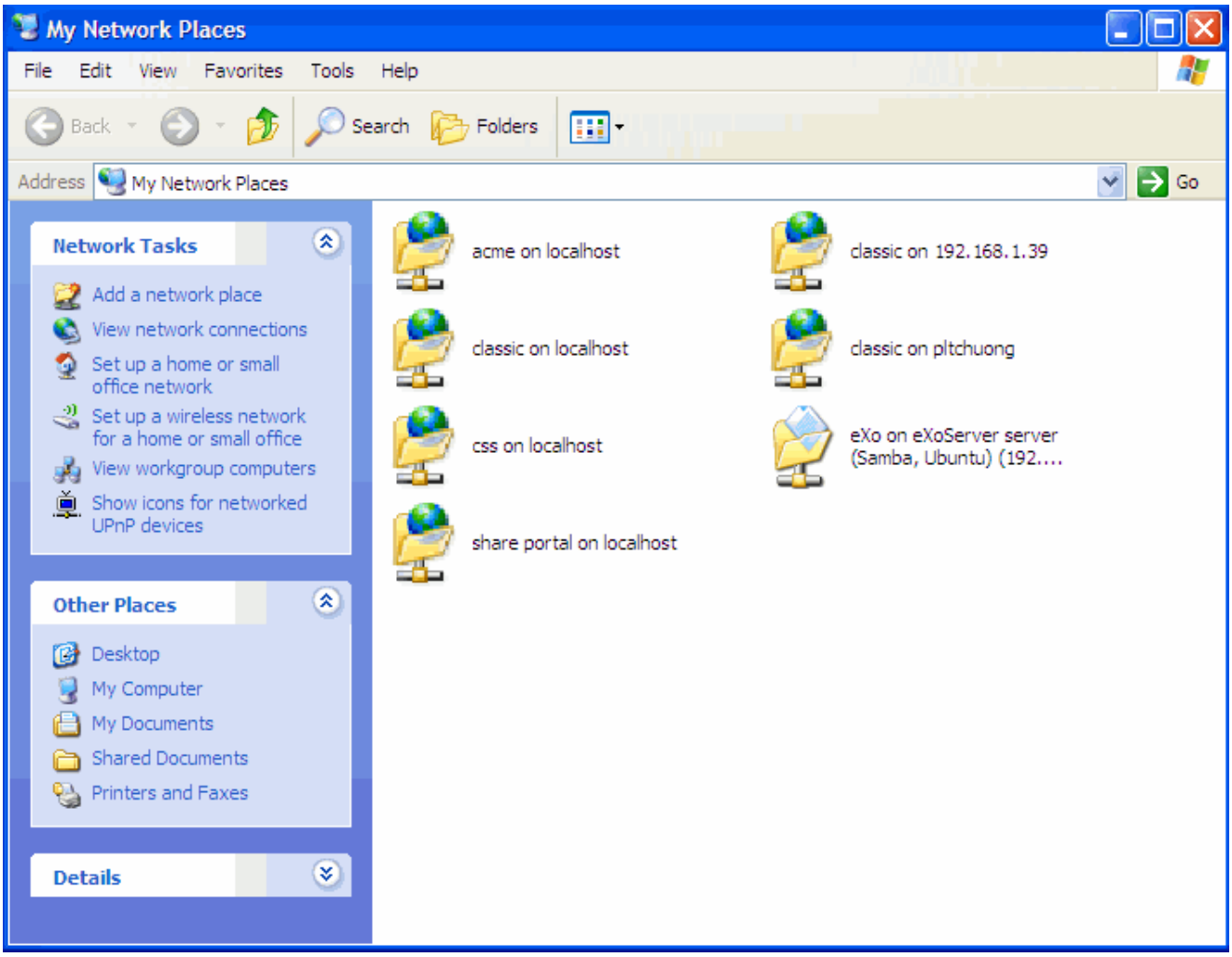
Now, we can do with our sites by FrontPage and Dreamweaver.

3.4. How to do with WebDAV in WCM

WCM supports you two ways to use WebDAV in sites management. Do as follows:

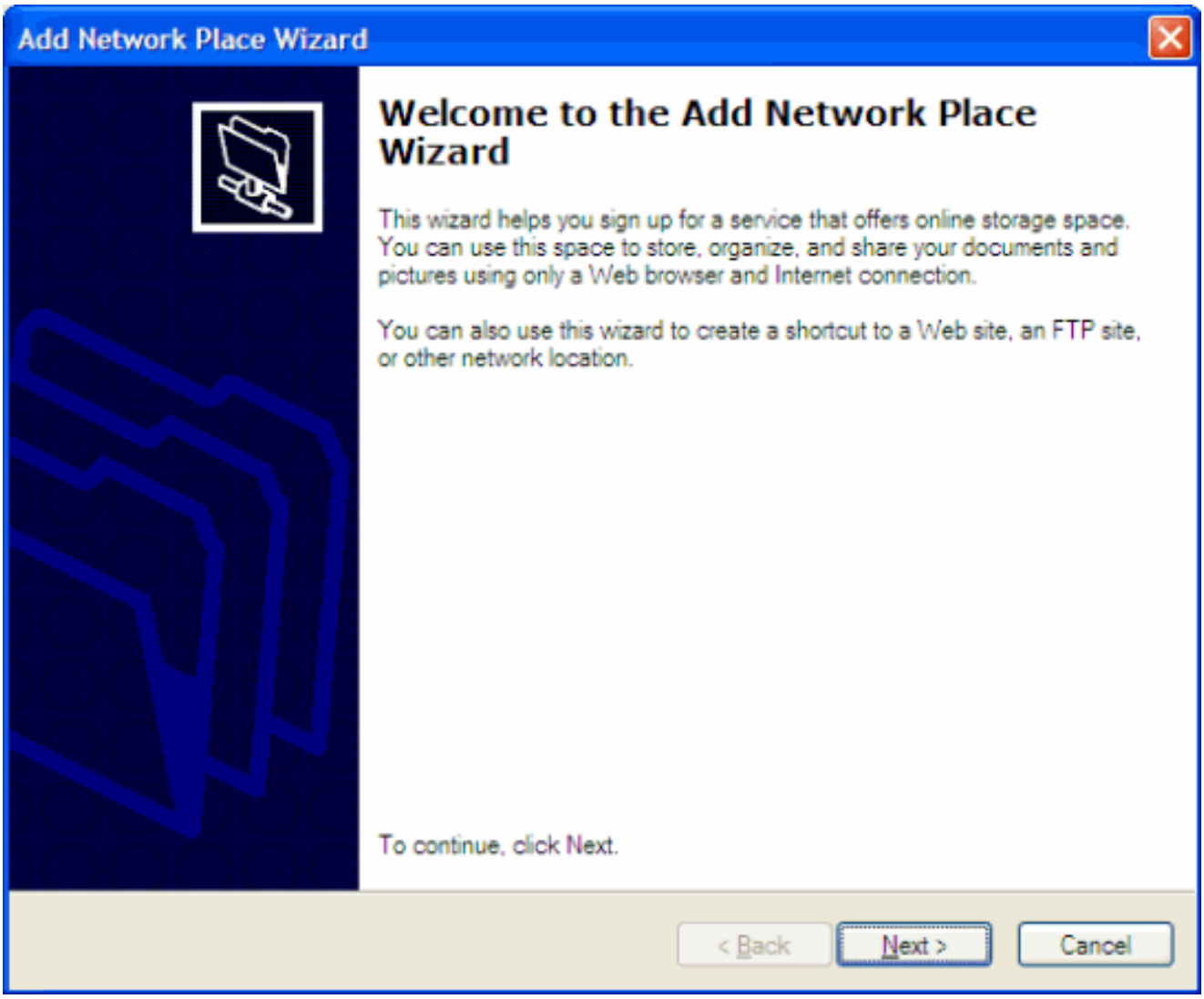
The 1st way : This way is used when you are on a Window System and your computer has to be connected to the Internet or Intranet.

1. Go into **My Network Places** folder on your local computer. You will see all shared files and folders like below:

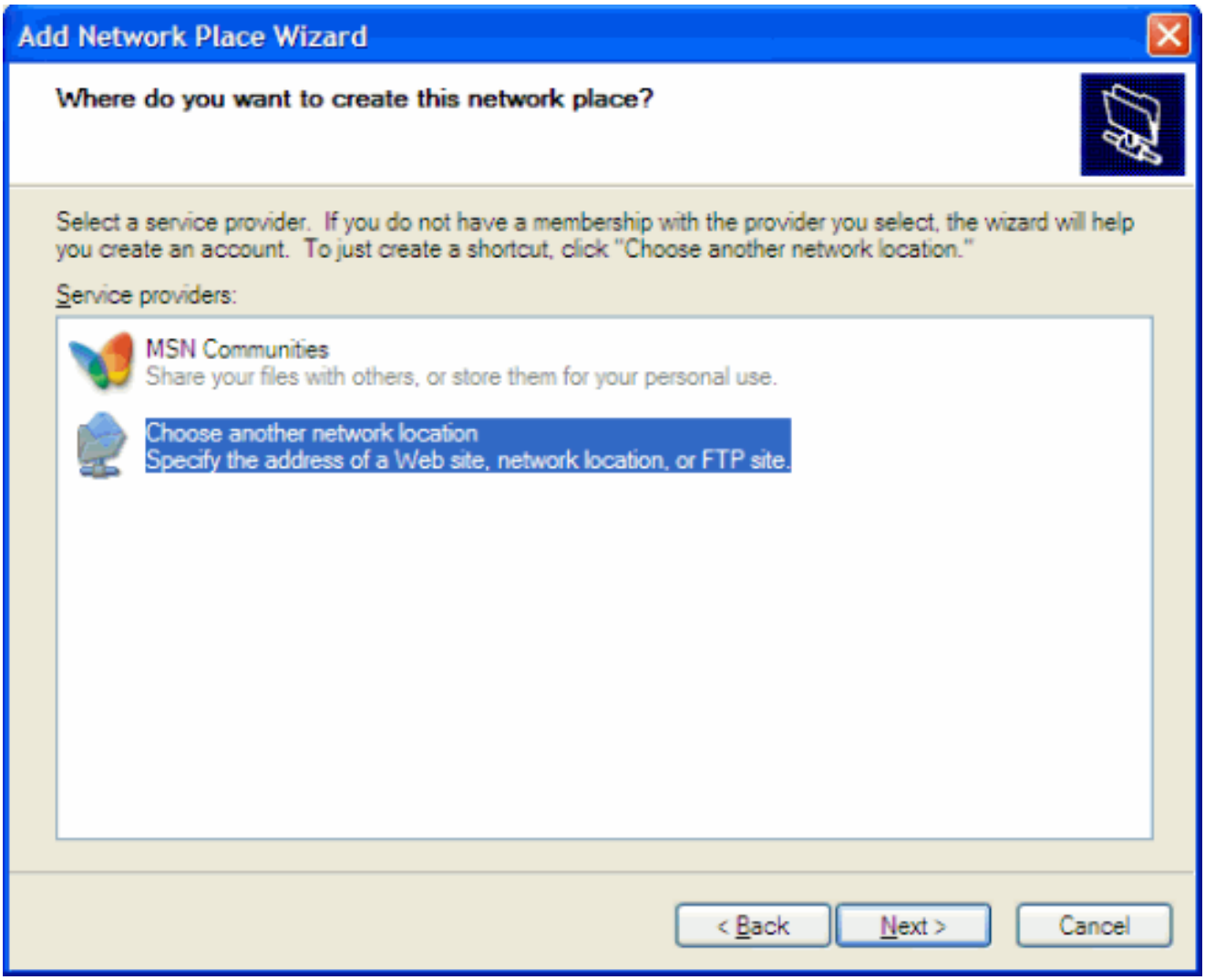


2. To show site list, do as following steps to use WebDAV:

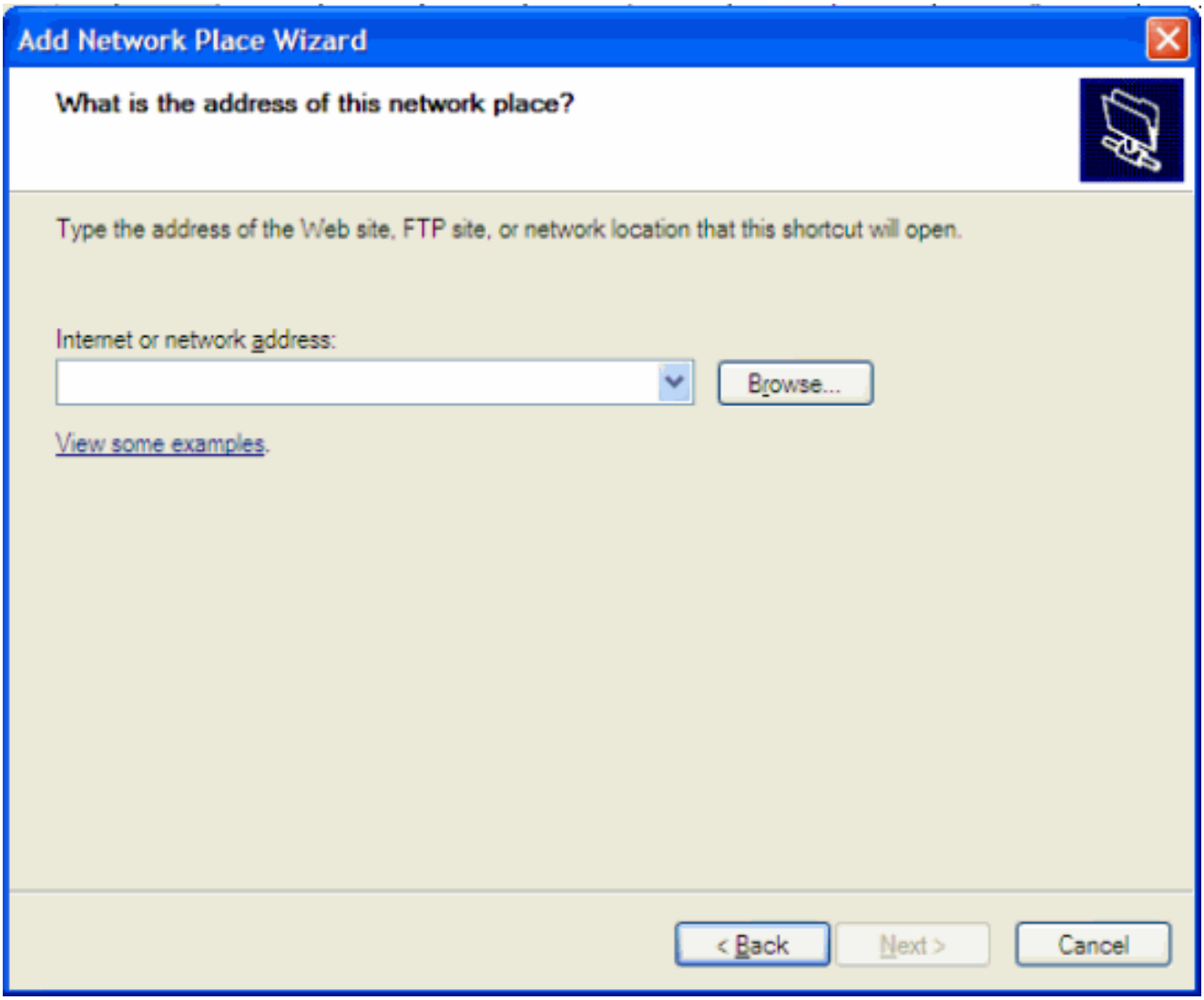
- Click 'Add a network place' link at the left to open Add Network Place Wizard form:



- Click **Next** button to open form to choose the location that you want to create the network place:

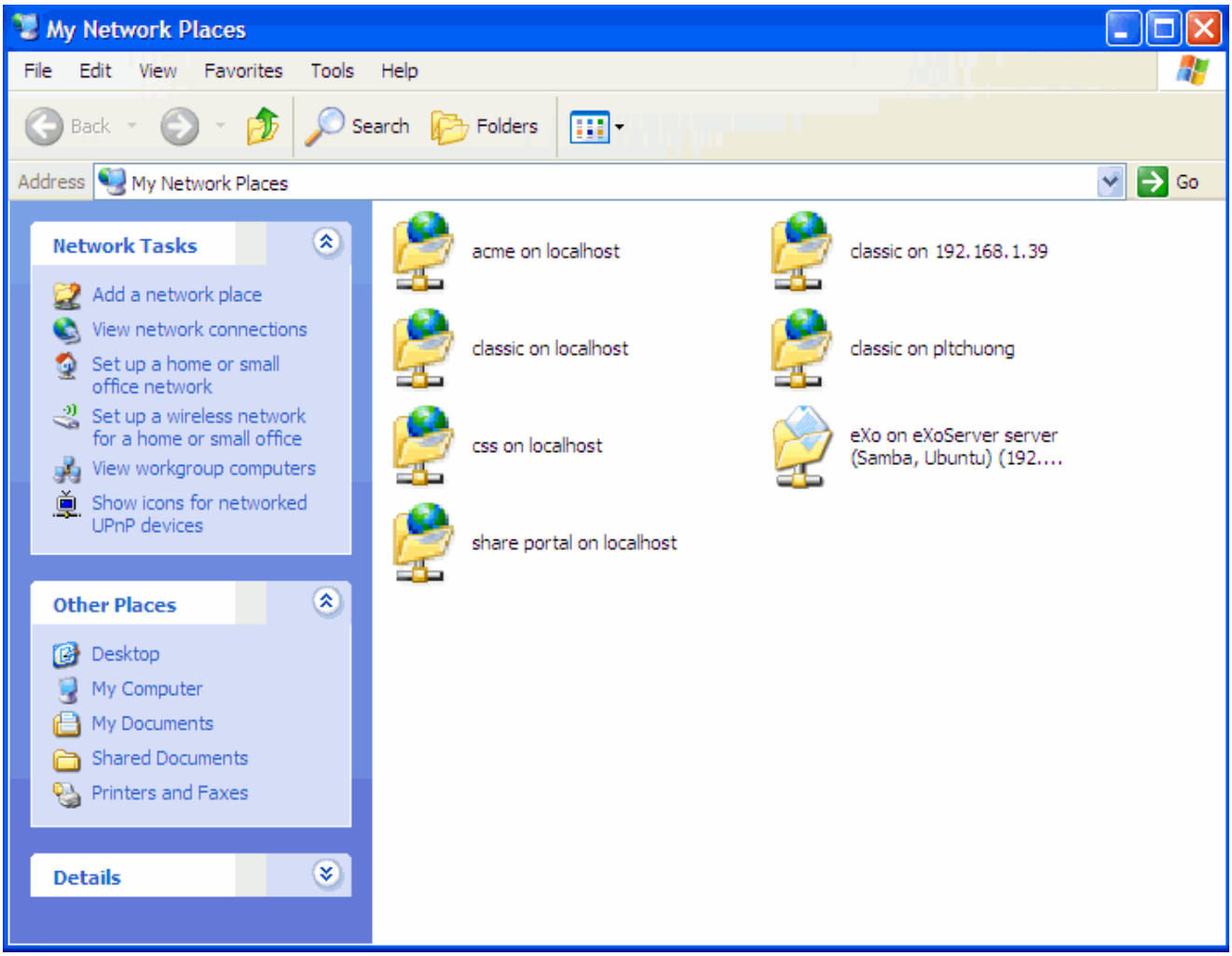


- Select 'Choose another network location' to just create a shortcut. Then you will be asked to enter an internet or a network address to refer.



- Enter or copy a link to view a site by WebDAV in a browser into 'Internet or network address' field. For an example, the link to view 'acme' site by WebDAV is: http://localhost:8080/portal/rest/private/jcr/repository/collaboration/sitescontent/live/acme

- Click **Next** button and wait some minutes to take affect. A folder named 'acme on localhost ' is corresponding to the inputted link will appear in **My Network Places** folder like below:



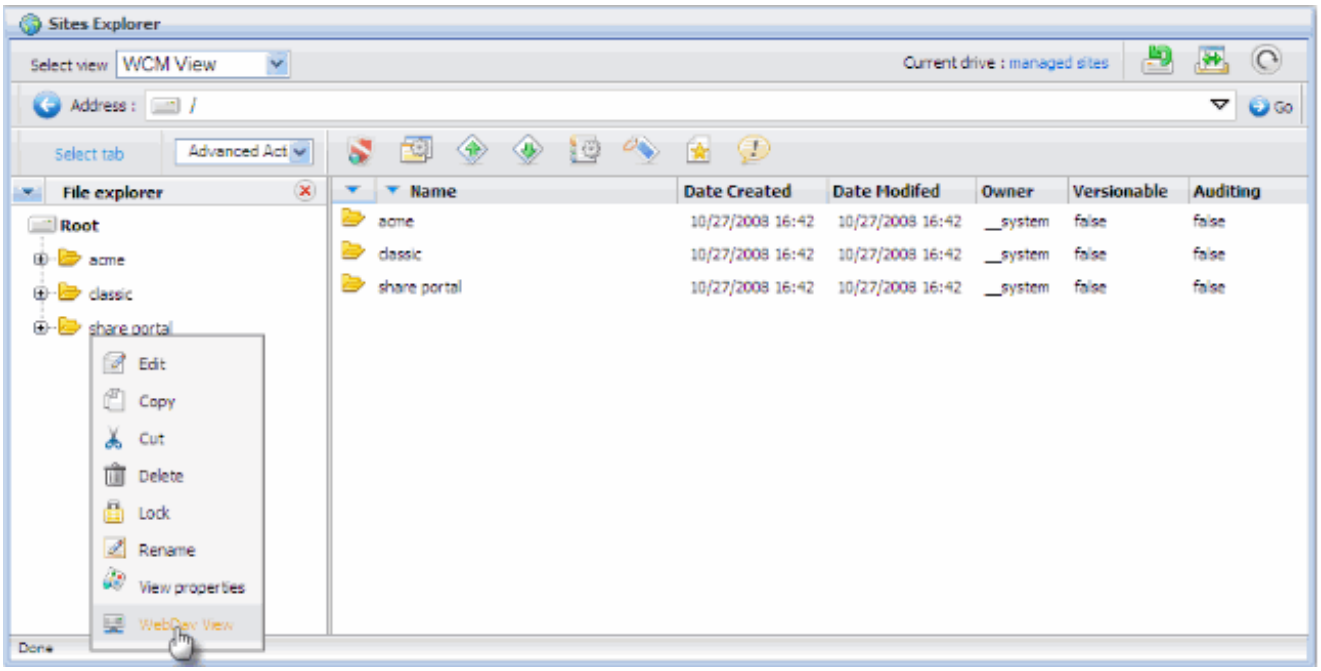
- Go into the folder corresponding to the site which you want to do actions on it.

The 2nd way:* + 1.Using IE browser to access http://localhost:8080/portal

2.Go to **Sites Content Management** --> **Sites Explorer** page.

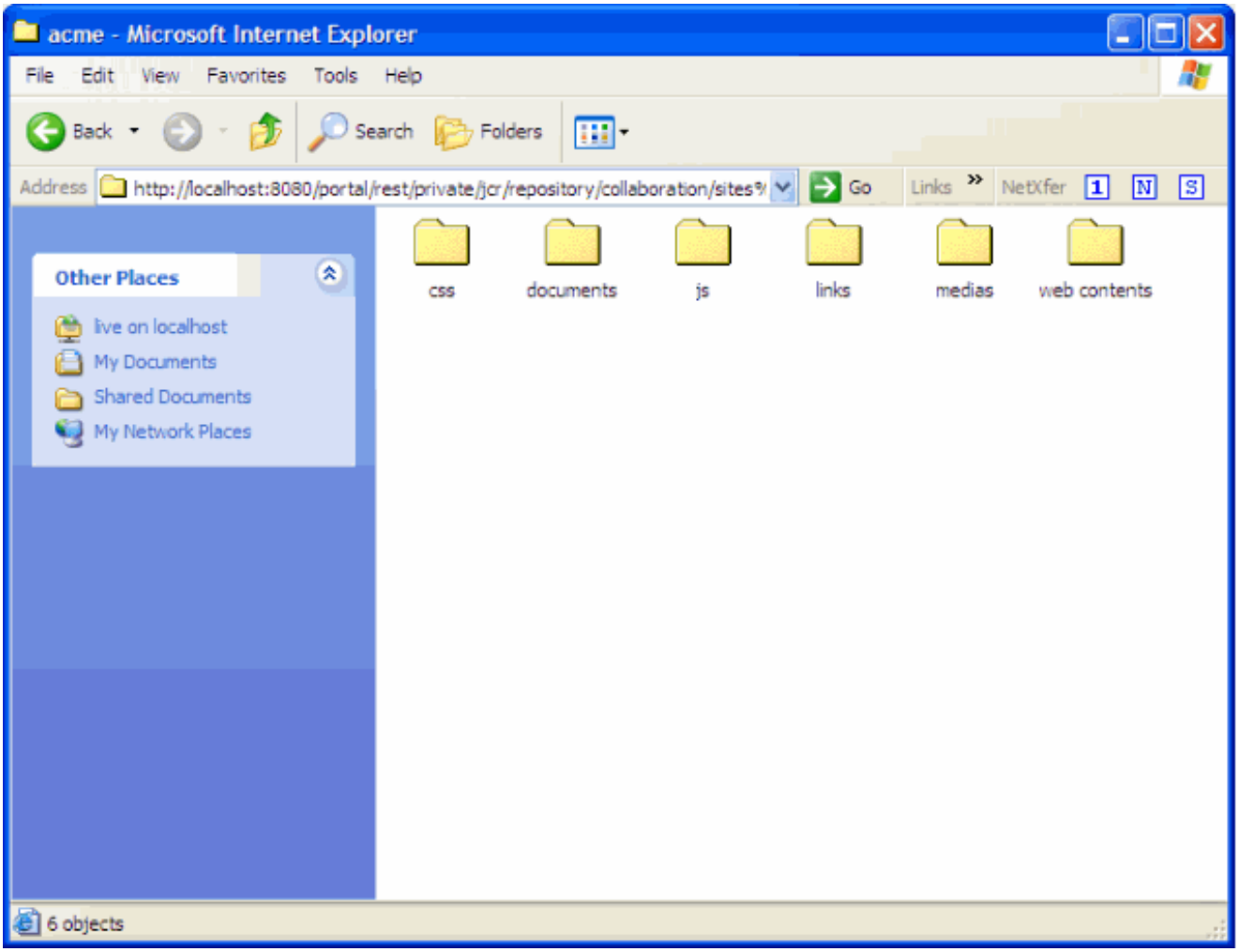
3.Go into **managed sites** drive by selecting this drive. You will see all sites on the left pane like illustration below.

4.Right - click on a site that you want to view by WebDAV and select 'WebDAV View' function in menu like:



5.The selected site will be viewed by WebDAV like illustration below. In which, each web content is a folder with default folders in it.

In here, an user can do with WebDAV as do with Window:



- An user can copy a HTML file and then paste in **Web Content** Folder to create new web content.

- An user can copy a CSS file on **CSS** folder to change stylesheet for site.
- An user can copy a JS file in **JS** folder.
- An user can copy some documents and then paste on **Documents** folder.

Chapter 4. Configuration

4.1. Javascript for Navigation and Sitemap

4.1.1. Navigations Object

You always have a JSON object of navigation: eXo.env.portal.navigations. It contains all data of site's navigation.

-
  - Each portal has 3 types of navigation: Portal navigation, Group navigation and User navigation. Thus navigation object has 3 childrens.
  - Each type of navigation has nodes property, it contains all navigation nodes of that navigation type.
  - Each navigation node has following properties: uri (String) : Navigation node's URI. icon (String): Navigation node's icon. visible (Boolean): A navigation node can be visible or invisible. If visible, it can be shown in navigation. resolvedLabel (String): Navigation node's label. name (String): Navigation node's name. children (Object): Navigation node's children.

4.1.2. Get Current Navigation Node Function

getCurrentNavigationNode(navigations, selectedNodeUri) function:

You also have a function to get selected navigation node object. In this function, navigations is eXo.env.portal.navigations, and selectedNodeUri is an URI of selected navigation node, you can get it from eXo.env.portal.selectedNodeUri (e.g. "/home/yourPage"). You can use this function to change some styles for the selected navigation node.

4.1.3. Navigation or Sitemap Creation

Define your own **renderNavigations()** function or **renderSitemap()** function by using **eXo.env.portal.navigations** and put it in default.js of navigation web content.

4.2. JavaScript for Breadcrumbs

4.2.1. Breadcrumb Object

This Breadcrumb object contains two properties:

- **resolvedLabel**: name of navigation node in current language or title of current document.
- **uri**: navigation node uri

For example:

```
JsonObj = {
```

```
          resolvedLabel: administration,
          uri: "#"
        }
```

4.2.2. Get Breadcrumbs() Function

This function is used to get an array of breadcrumb object.

4.3. JavaScript for Search Box

4.3.1. quickSearch(resultPageURI) function

- This function will be redirected to the page which shows search results by setting url of a page for **window.location** variable.
- Accept one parameter **resultPageURI** to indicate the correct page to be used to view results. If customers don't pass the value for **resultPageURI**, we will set a default value is **searchResult**.
- url of result page in well format: **http:/host-name]:port-number]/.../resultPageURI]**

For example: http://localhost:8080/portal/private/classic/eXoResult?portal=classic&keyword=exoplatform

4.3.2. quickSearchOnEnter(event, resultPageURI) function

- This function will call **quickSearch(resultPageURI)** after you press enter key in case textbox contains your keyword.
- Accept event and **resultPageURI** parameters, so we can check right when the enter key is pressed. If true (the enter key is pressed) it's very simple to call **quickSearch(resultPageURI)**, otherwise it will do nothing.
- We check a key that is pressed by **getKeynum(event)** function, it returns a number. If this number equals to 13 (keycode of enter key), enter key is pressed.

4.3.3. keepKeywordOnBoxSearch() function

- When you enter your keyword and click search or press enter key on textbox, the keyword is lost - it's not useful. This function is provided to make good one's shortcomings - keep your keyword that you've just entered to search.
- This function always be active when a web site is loaded. By using the function:

eXo.core.Browser.addOnLoadCallback(param1, param2) - you can have a look at this function in **exo.portal.web.eXoResources.Browse.js**. Here is an example:

```
eXo.core.Browser.addOnLoadCallback("keepKeywordOnBoxSearch", keepKeywordOnBoxSearch);
```

- How to get keyword for keeping. Firstly, I get a query string (it's value of **location.search**) and then paste it to get a value of keyword parameter. At last I store the keyword in textbox.

4.4. Content Wizard Size Configuration

4.4.1. Overview

Content edition in Single Content Viewer (SCV) portlet allows you easy to create or select a webcontent, document to present in a webpage. In our wcm, the content edition of SCV is wizard-oriented design and the wizard is displayed in a popup window. To let you choose the size of popup window to fit your content purpose and screen resolution, we defined the size of popup in configuration file as a service.

4.4.2. Configuration

The configuration file is under: ../wcmportal/.../WEB-INF/conf/wcm/webui-properties-configuration.xml(in source code) and look like following code snippet:

```
<configuration>
   <component>
    <key>org.exoplatform.wcm.webui.WebUIPropertiesConfigService</key>
    <type>org.exoplatform.wcm.webui.WebUIPropertiesConfigService</type>
    <init-params>
              <properties-param>
               <name>SCV.popup.size.in.edit.portlet.mode</name>
               <description>size of popup in edit mode of portlet</description>
               <property name="width" value="850"/>
               <property name="height" value="525"/>
              </properties-param>
              <properties-param>
                 <name>SCV.popup.size.in.quickdedit</name>
                 <description>size of popup in edit mode of portlet</description>
                 <property name="width" value="950"/>
                 <property name="height" value="750"/>
             </properties-param>
    </init-params>
  </component>
</configuration>
```

- To change the size of popup window when editing content in **page creation wizard** or in edit mode of portlet, please change value of properties: **SCV.popup.size.in.edit.portlet.mode**.

- To change the size of popup window when editing content by click on **quick edit action**, please change value of properties: **SCV.popup.size.in.quickdedit**

4.5. Content Classification

4.5.1. Overview

In a content management system, one of main benefits that it supports content classification framework (aka: content sorter)to allow automatically classifying your content(document, webcontent, video, audio...) into predefined location that you want and belong to some content's attributes like: author, type of content, datetime, name of content, etc.,

4.5.2. Implementation

The implementation of this feature is in research and development. Here the some idea for that:

- Each type of classify is a component plugin.
- An classify service is created to manage the classify plugin.
- The classify plugin can be used by two ways: from configuration or as a DMS actions.

The NodeClassifyService look like following code snippet:

```
public interface NodeClassifyService {

  public void addClassifyPlugin(ComponentPlugin componentPlugin) throws Exception;
  public Collection<NodeClassifyPlugin> getAllClassifyPlugins() throws Exception;
  public NodeClassifyPlugin getNodeClassifyPlugin(String type) throws Exception;
  public void removeClassifyPlygin(String type) throws Exception;
  public <T extends NodeClassifyPlugin> T getNodeClassifyPluginByType(Class<T> clazz) throws Exception;

}
```

The NodeClassifyPlugin look like following code snippet:

```
public abstract class NodeClassifyPlugin implements ComponentPlugin {

  private String name;
  private String desc;

  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getDescription() { return desc; }
  public void setDescription(String desc) { this.desc = desc; }
  public abstract void classifyChildrenNode(Node parent) throws Exception;
}
```

To configure the plugin:

```
<component>
     <key>org.exoplatform.services.jcr.ext.classify.NodeClassifyService</key>
     <type>org.exoplatform.services.jcr.ext.classify.impl.NodeClassifyServiceImpl</type>
     <component-plugins>
            <component-plugin>
                        <name>AlphabetClassifyPlugin</name>
                        <set-method>addClassifyPlugin</set-method>
                        <type>org.exoplatform.services.jcr.ext.classify.impl.AlphabetClassifyPlugin</type>
                  </component-plugin>
            <component-plugin>
                        <name>DateTimeClassifyPlugin</name>
                        <set-method>addClassifyPlugin</set-method>
                        <type>org.exoplatform.services.jcr.ext.classify.impl.DateTimeClassifyPlugin</type>
                        <init-params>
                     <properties-param>
                       <name>plugin-params</name>
```

```
                        <description>service-params</description>
                        <property name="DateTimeTemplate" value="YYYY-#{YYYY+4}/MM"/>
                        <property name="DateTimePropertyName" value=""/>
                        <property name="StartTime" value="2002-2"/>
                        <property name="EndTime" value="2020-9"/>
                      </properties-param>
                         </init-params>
                </component-plugin>
          <component-plugin>
                         <name>TypeClassifyPlugin</name>
                         <set-method>addClassifyPlugin</set-method>
                         <type>org.exoplatform.services.jcr.ext.classify.impl.TypeClassifyPlugin</type>
                </component-plugin>
          </component-plugins>
   </component>
```

4.6. Deploy Single Content to a Predefined Page

4.6.1. Overview

To build up a new website, you need to prepare a lot of contents for displaying when your site is put in production for the first time. This tutorial help you create and deploy your single content(a webcontent or DMS document) to a predefined web-page with Single Content Viewer in our eXo WCM.

4.6.2. Content Creation and Exporting to xml file

With our wcm, we support several ways to create your content(document or webcontent):

- WCM:Creating Web Content using the Create Page Wizard
- Create web content by single content viewer(SCV)
- Create web content in sites explorer
- Create web content by using WebDAV

When your content is created by these ways, you can see the content in 2 locations in SiteExplorer

- Under Docucments folder or its childen of site content storage location
- Or under Web contents folder of its childen of the site content storge

(Please read our about Site Content Storage for more details).

After you have the content, please export it as xml document view by export feature. Here is screenshot to export Service and Products web-content in our acme site(Note that you can do the same step for other document type)

Note

Disable the publication-service config in configuration.xml *before you create* the site artifacts:

war:/conf/wcm/publication-configuration.xml To disable it, you have to comment those external component plugins : StateAndVersionPublicationHandler,CmsService.event.postCreate,CmsService.event.postEdit,PortalArtifactsIn

Then export the node(s).

Don't forget to re-enable this configuration file!

Of course you can use different computers for creation and deployment, so that you don't have to reconfigure and restart WCM.

4.6.3. Content Deploying Configuration

This step is provided to configure so that you can deploy your content to right location that be created. After you have your content in xml document view version, you need configure to re-deploy it to acme/web contents folder when your web server application run at the first time. To do that, put your xml file in your portal war and configure for ContentInitializerService to initiate the content.

```
<external-component-plugins>
          <target-component>org.exoplatform.services.deployment.ContentInitializerService</target-component>
      <component-plugin>
        <name>Content Initializer Service</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.deployment.plugins.XMLDeploymentPlugin</type>
        <description>XML Deployment Plugin</description>
        <init-params>
          <object-param>
            <name>ACME Servives And Products data</name>
            <description>Deployment Descriptor</description>
            <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
              <field  name="target">
                      <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
                              <field  name="repository"><string>repository</string></field>
                              <field  name="workspace"><string>collaboration</string></field>
                              <field  name="nodePath"><string>/sites content/live/acme/web contents</string></fi
                      </object>
              </field>
              <field  name="sourcePath"><string>war:/conf/wcm/artifacts/site-resources/acme/contents/ServicesAnd
            </object>
          </object-param>

        </init-params>
      </component-plugin>
   </external-component-plugins>
```

With this configuration, the **ServicesAndProducts.xml** file is located **sourcePath** params and will be imported to **Target** location is in: **"repository"** repository, **"collaboration"** workspace and under folder **"/sites content/live/acme/web contents"** (full path for web contents in acme site content storage)

1 Content Association with a Single Content Viewer

The last step is to associate your content with a Single Content View in a page. Define the single content viewer in page.xml that is used to present the content like that (the name of the SCV is **"<instance-id>portal#acme:/web-presentation/SingleContentViewer/acme-serpro</instance-id>"**)

```
<page>
          <page-id>portal::acme::homepage</page-id>
    <owner-type>portal</owner-type>
    <owner-id>acme</owner-id>
    <name>homepage</name>
    <title>Home Page</title>
          <access-permissions>Everyone</access-permissions>
          <edit-permission>*:/platform/administrators</edit-permission>
```

```
          <container id="ACMEHomepage" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
                       <container id="ACMEContainer" template="system:/groovy/wcm/webui/container/UITableColumn
                         <container id="ACMESerProContainer" template="system:/groovy/portal/webui/container/UI
                                 <application>
                                   <instance-id>portal#acme:/web-presentation/SingleContentViewer/acme-serpro</
                                   <title>Services and Products</title>
                                 </application>
                         </container>
                 </container>
          </container>
  </page>
```

And configure your portlet(in portlet-preference.xml file) points to the content that be created like that:

```
<portlet-preferences>
    <owner-type>portal</owner-type>
    <owner-id>acme</owner-id>
    <window-id>portal#acme:/web-presentation/SingleContentViewer/acme-serpro</window-id>
    <preference>
      <name>repository</name>
      <value>repository</value>
      <read-only>false</read-only>
    </preference>
    <preference>
      <name>workspace</name>
      <value>collaboration</value>
      <read-only>false</read-only>
    </preference>
    <preference>
      <name>nodeIdentifier</name>
      <value>/sites content/live/acme/web contents/Services And Products</value>
      <read-only>false</read-only>
    </preference>
    <preference>
      <name>ShowQuickEdit</name>
      <value>true</value>
      <read-only>false</read-only>
    </preference>
  </portlet-preferences>
```

And here is final result you can see on acme homepage:

4.7. Web Content Management Cache : Configuration and Management

Starting with release 1.2.2 of eXo WCM a cache has been added between the eXo JCR and the WCM. This cache is part of the WCM Composer Service in charge of delivering content to the viewer portlets (Single Content, Content List, ...) and avoiding hereby access to the JCR. This cache is based on eXo Cache, and it is a simple FIFO cache.

4.7.1. Cache level

Each user has an individual cache. Furthermore there is one cache for the anonymous user which is applied for all visitors of the web site (those who did not log in). In public websites most users surf as anonymous users therefore this user is also called *public user*.

Each time a content is published, the cache of all anonymous users (public cache) and the cache of the publisher himself is refreshed. This means that the cache of other users is not immediately invalidated.

It is possible to configure or to deactivate the cache in the publication configuration file. This file is located in:

$EXO*CONF/wcm/publication-configuration.xml*

For example in the default eXo Tomcat bundle this file is located in $TOMCAT*HOME/webapps/portal/WEB-INF/conf/wcm/publication-configuration.xml*

```
<component>
  <key>org.exoplatform.services.wcm.publication.WCMComposer</key>
  <type>org.exoplatform.services.wcm.publication.WCMComposerImpl</type>
  <init-params>
    <value-param>
      <name>useCache</name>
      <value>true</value>
    </value-param>
  </init-params>
</component>
```

You can choose to use the cache or not - the default value of *useCache* is *true*. Note that with an enabled cache, you may perceive some delay between content publication and website front-end update.

```
<external-component-plugins>
  <target-component>org.exoplatform.services.cache.CacheService</target-component>
  <component-plugin>
    <name>addExoCacheConfig</name>
    <set-method>addExoCacheConfig</set-method>
    <type>org.exoplatform.services.cache.ExoCacheConfigPlugin</type>
    <description>Configures the cache for query service</description>
    <init-params>
            <object-param>
              <name>cache.config.wcm.composer</name>
              <description>The default cache configuration</description>
              <object type="org.exoplatform.services.cache.ExoCacheConfig">
                <field name="name"><string>wcm.composer</string></field>
                <field name="maxSize"><int>300</int></field>
                <field name="liveTime"><long>600</long></field>
                <field name="distributed"><boolean>false</boolean></field>
                <field name="implementation"><string>org.exoplatform.services.cache.concurrent.ConcurrentFIFOE
              </object>
            </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

You can modify several parameters to control the behavior of your WCM instance.

- maxSize: The maximum number of elements in your cache
- liveTime: The amount of time (in milliseconds) an element is not written or read before it is evicted.
- distributed: Indicates if the cache is distributed
- implementation: FIFO cache is default, in theory you might use a different implementation

4.7.2. Invalidate the cache manually

eXo WCM Cache is based on eXo Cache. The eXo Cache is manageable by JMX. You can use any JMX application and access the MBean called:

```
exo:service=cache,"name=wcm.composer"
```

To invalidate the cache you need to call the clearCache() operation. This operation deletes the content of all WCM caches. This means that at the next request of a user the content of a portlet is retrieved from the JCR and the user's individual cache refilled (see above explanations about individual caches).

4.7.3. Cache and Front office

eXo WCM Cache is used in almost all eXo WCM's presentation portlets, this means: Single Content Viewer (SCV), Content List Viewer (CLV), Parameterized Content Viewer (PCV), Parameterized Content List Viewer (PCLV). In these portlets, after the content is updated, we have to wait some time (the time is configured) until the cache is synchronized with new data.

Currently, eXo WCM support 2 modes to display contents, to help Administrators and Managers.

- Live mode: Use cached contents.
- Edit mode: Use real contents.

Chapter 5. DMS

Chapter 6. Configuration

6.1. Content Browser Portlet Preferences

For now, ECM supported four usecases to modify Content Browser portlet:

- Path - Query - Script - Detail document

6.1.1.1. Usecase is Path

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-preferences>
  <!-- Specify the usecase will be use -->
  <preference>
    <name>usecase</name>
    <value>path</value>
    <read-only>false</read-only>
  </preference>
  <!-- Setup the default repository name for content browser portlet -->
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the workspace name will be use in repository -->
  <preference>
    <name>workspace</name>
    <value>system</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the root path and you will see all child nodes when browse the portlet -->
  <preference>
    <name>path</name>
    <value>/jcr:system/exo:ecm/exo:taxonomies</value>
    <read-only>false</read-only>
  </preference>
  <!--
          When you choose usecase is path you can use this preference to allow show reference
          documents or not
  -->
  <preference>
    <name>reference</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          When you choose usecase is path you can use this preference to allow show child node
          is document or not
  -->
  <preference>
    <name>child</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This one will allow show the tag map or not.
  -->
  <preference>
    <name>viewTagMap</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          When you choose usecase is any type you should specify the template type which will be show
          in the UI
          Check for detail in ecm-views-configuration.xml
          For now, ECM supported two type are PathList & TreeList
  -->
  <preference>
    <name>template</name>
```

```
    <value>PathList</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference related to template, this one specify the template which will be use to
          display child node document.
  -->
  <preference>
    <name>boxTemplate</name>
    <value>DocumentView</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use when choose usecase is path or query. This one specify the
          number of item for each page.
  -->
  <preference>
    <name>nbPerPage</name>
    <value>20</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use when choose usecase is path. This one allow show or not the
          tool bar.
  -->
  <preference>
    <name>viewToolbar</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>

  <!--
          This preference will be use when choose any usecase. This one allow show or not the comment action.
  -->
  <preference>
    <name>viewComment</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use when choose any usecase. This one allow show or not the vote action.
  -->
  <preference>
    <name>viewVote</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

6.1.1.2. Usecase is Query

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-preferences>
  <!-- Specify the usecase will be use -->
  <preference>
    <name>usecase</name>
    <value>query</value>
    <read-only>false</read-only>
  </preference>
  <!-- Setup the default repository name for content browser portlet -->
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the workspace name will be use in repository -->
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <!--
          When you choose usecase is query you should use this preference to specify the status
          of query is new query or existing query.
```

```
-->
<preference>
  <name>isAddNew</name>
  <value>true</value>
  <read-only>false</read-only>
</preference>
<!--
        When you choose usecase is query you have to use this preference to setup the type
        of query will be use is xpath or sql
-->
<preference>
  <name>queryLanguage</name>
  <value>xpath</value>
  <read-only>false</read-only>
</preference>
<!--
        You have to use this preference to init the query statement when your query status is new
-->
<preference>
  <name>queryStatement</name>
  <value>/jcr:root/Documents/Live//element(*, exo:article)</value>
  <read-only>false</read-only>
</preference>
<!--
        You should specify the template type which will be show in the UI
        For now, ECM support one type is QueryList.
-->
<preference>
  <name>template</name>
  <value>QueryList</value>
  <read-only>false</read-only>
</preference>
<!--
        If your query status is existing then you have to use this
        preference to specify the query type is personal or shared query.
-->
<preference>
  <name>queryType</name>
  <value></value>
  <read-only>false</read-only>
</preference>
<!--
        When you choose usecase is query and query status is existing then you have to use this
        preference to specify the query location.
        Normaly all existing shared queries stored in folder: /jcr:system/exo:ecm/queries
        personal queries stored in folder: /Users/$username/Private/Queries
-->
<preference>
  <name>queryStore</name>
  <value></value>
  <read-only>false</read-only>
</preference>
<!--
        This preference related to template, this one specify the template which will be use to
        display child node document.
-->
<preference>
  <name>boxTemplate</name>
  <value>DocumentView</value>
  <read-only>false</read-only>
</preference>
<!-- This one specify the number of item for each page. -->
<preference>
  <name>nbPerPage</name>
  <value>20</value>
  <read-only>false</read-only>
</preference>
<!-- This preference will allow show the tool bar or not. -->
<preference>
  <name>viewToolbar</name>
  <value>true</value>
  <read-only>false</read-only>
</preference>
<!--
        This preference will be use when choose any usecase except detail-document.
        This one allow show the tag map or not.
-->
<preference>
```

```
    <name>viewTagMap</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use when choose any usecase. This one allow show or not the comment action.
  -->
  <preference>
    <name>viewComment</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use when choose any usecase. This one allow show or not the vote action.
  -->
  <preference>
    <name>viewVote</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

6.1.1.3. Usecase is Script

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-preferences>
  <!--
          Specify the usecase will be use
  -->
  <preference>
    <name>usecase</name>
    <value>script</value>
    <read-only>false</read-only>
  </preference>
  <!-- Setup the default repository name for content browser portlet -->
  <preference>

    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the workspace name will be use in repository -->
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <!--
    You have to choose one script which will be executed when the porlet init.
  -->
  <preference>
    <name>scriptName</name>
    <value>GetDocuments.groovy</value>
    <read-only>false</read-only>
  </preference>
  <!--
          Specify the template type which will be show in the UI
                For now, ECM supported one sample type is ScriptList
        -->
  <preference>
    <name>template</name>
    <value>ScriptList</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference related to template, this one specify the template which will be use to
          display child node document.
  -->
  <preference>
    <name>boxTemplate</name>
    <value>DocumentView</value>
    <read-only>false</read-only>
  </preference>
  <!-- This one specify the number of item for each page. -->
```

```
  <preference>
    <name>nbPerPage</name>
    <value>20</value>
    <read-only>false</read-only>
  </preference>
  <!-- This preference will be use to allow show the tag map or not -->
  <preference>
    <name>viewTagMap</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!-- This preference will be use to allow show the comment action or not. -->
  <preference>
    <name>viewComment</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <!-- This preference will be use to allow show the vote action or not. -->
  <preference>
    <name>viewVote</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

6.1.1.4. Usecase is Detail document

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-preferences>
  <!--
          Specify the usecase will be use
  -->
  <preference>
    <name>usecase</name>
    <value>detail-document</value>
    <read-only>false</read-only>
  </preference>
  <!-- Setup the default repository name for content browser portlet -->
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the workspace name will be use in repository -->
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the category path which contain the documents -->
  <preference>
    <name>path</name>
    <value>/Documents/Live</value>
    <read-only>false</read-only>
  </preference>
  <!-- Specify the name of document in category when you choose the usecase is detail-document -->
  <preference>
    <name>documentName</name>
    <value>article</value>
    <read-only>false</read-only>
  </preference>
  <!--
          This preference will be use specify the template which will be use to
          display document.
  -->
  <preference>
    <name>boxTemplate</name>
    <value>DocumentView</value>
    <read-only>false</read-only>
  </preference>
  <!-- This preference will be use to allow show the comment action or not. -->
  <preference>
    <name>viewComment</name>
    <value>true</value>
```

```
    <read-only>false</read-only>
  </preference>
  <!-- This preference will be use to allow show the vote action or not. -->
  <preference>
    <name>viewVote</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

6.2. Build your own Metadata Extraction

ECM is capable of automatically extracting metadata from files at upload time. We have extractors for most common office documents (.doc, .pdf, .ppt, .xls, ...).

But this mechanism is extensible through plugins for your own metadata management. This tutorial will show you how to extract the metadata from a simple properties file.

Note

For this tutorial, you will need java 5, maven 2 and eclipse

6.2.1. Create and deploy my extractor

- Create a class called **org.exoplatform.tutorial MyMetadataExtractor** which has to extend to **org.exoplatform.services.document.impl.BaseDocumentReader**. As follow:

```
import org.exoplatform.services.document.impl.BaseDocumentReader;

public class MyMetadataExtractor extends BaseDocumentReader {
```

- Implement the **getContentAsText** methods which are dedicated to the full content extraction. As follow:

```
/**
 * Text extraction for the full text indexing
 */
public String getContentAsText(InputStream input) throws Exception {
        // Create a Properties Object
        Properties properties = new Properties();
        // Load the properties from the input stream
        properties.load(input);
        // Create a StringBuilder Object to append the full content of my file
        StringBuilder content = new StringBuilder();
        for (Enumeration keys = properties.keys(); keys.hasMoreElements(); ) {
                String key = (String) keys.nextElement();
                // Get the value from the key
                String value = properties.getProperty(key);
                // Append the value to the content
                content.append(value).append(' ');
        }
        return content.toString();
}

/**
 * Text extraction for the full text indexing with a specific character encoding
 */
public String getContentAsText(InputStream input, String encoding) throws Exception {
        return getContentAsText(input);
}
```

Note: This content will be indexed by eXo Platform in full text. It is easy to retrieve the file from its content through the **Search Component**.

- Add the configuration file to declare the component to eXo Platform, create a **configuration.xml** file in the conf/portal directory (into the source folder) and add the following contents:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
  <!- Define my Metadata Extractor ->
  <external-component-plugins>
    <target-component>org.exoplatform.services.document.DocumentReaderService</target-component>
    <component-plugin>
      <name>my.document.reader</name>
      <set-method>addDocumentReader</set-method>
      <type>org.exoplatform.tutorial.MyMetadataExtractor</type>
      <description>to read my specific stream</description>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

- Build the resulting jar by using maven(maven 2 must be correctly installed in your system), you can create a **pom.xml** in the root folder of the project with the following contents:

```
<project>
        <modelVersion>4.0.0</modelVersion>
        <groupId>org.exoplatform.tutorial</groupId>
        <artifactId>exo.tutorial.metadata-extraction</artifactId>
        <packaging>jar</packaging>
        <version>trunk</version>
        <description>Tutorial metadata extraction</description>
        <dependencies>
                <dependency>
                        <groupId>org.exoplatform.core</groupId>
                        <artifactId>exo.core.component.document</artifactId>
                        <version>trunk</version>
                        <scope>compile</scope>
                </dependency>
                <dependency>
                        <groupId>org.exoplatform.kernel</groupId>
                        <artifactId>exo.kernel.commons</artifactId>
                        <version>trunk</version>

                        <scope>compile</scope>
                </dependency>
                <dependency>
                        <groupId>org.exoplatform.kernel</groupId>
                        <artifactId>exo.kernel.container</artifactId>
                        <version>trunk</version>
                        <scope>compile</scope>
                </dependency>
        </dependencies>
        <build>
                <resources>
                  <resource>
                        <directory>src/main/java</directory>
                        <includes>
                          <include>**/*.xml</include>
                        </includes>
                  </resource>
                </resources>
        </build>
</project>
```

When you have done this, you can launch "mvn install" command from the root directory of the project. This

will create a jar file called **exo.tutorial.metadata-extraction-trunk.jar** into the target folder. **Note:** Ensure that the maven settings file use http://maven2.exoplatform.org/rest/maven2 as maven repository.

- Copy the jar into the *$TOMCAT**HOME}/lib** directory*
- Start tomcat and check for the following message "ExoContainer - org.exoplatform.tutorial.MyMetadataExtractor added to portal"

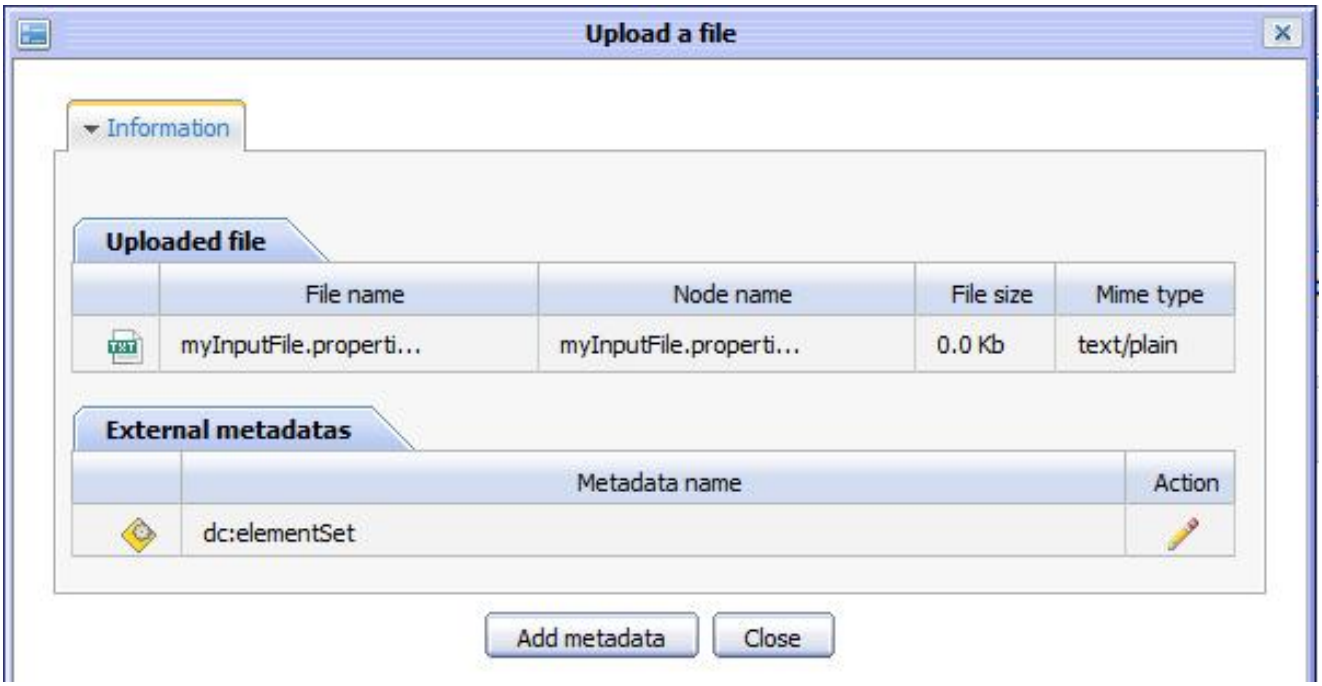
6.2.2. Test my extractor

To test the extractor we must:

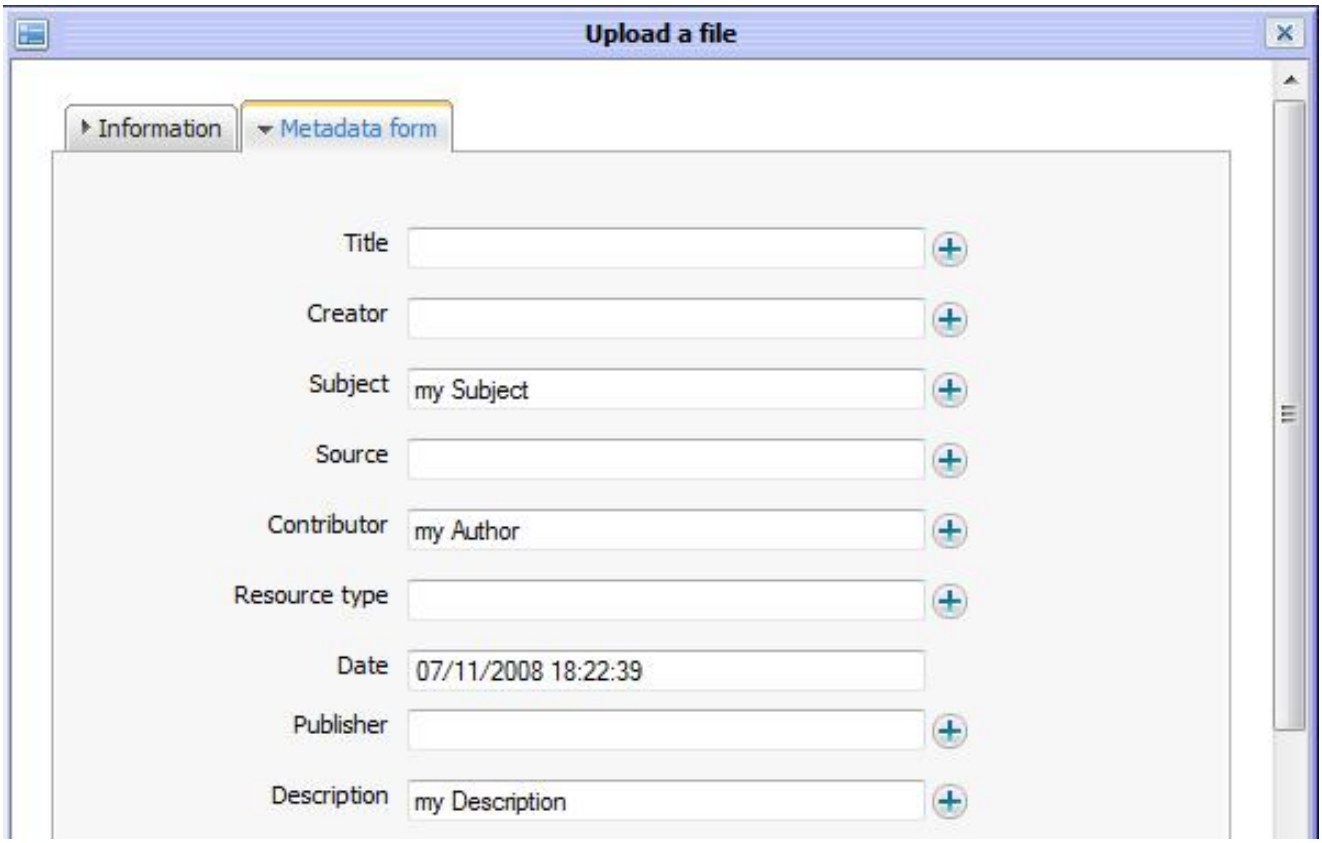
- Create a properties file called myInputFile.properties with the following contents:

```
Author=my Author
Subject=my Subject
Description=my Description
```

- Authenticate the portal as root
- Go to **Content Management** -> **File Explorer**
- Select a drive to upload the file like **Collaboration Center**
- Click on the **Upload** icon
- Choose the properties file that we created below then click on the **Upload** icon, the file will be uploaded to the server
- Click on the **Save** button, you will see the following window:



- Click on the **Edit** button to see the extracted metadata, you will see the following window:



- To easily retrieve your file you can select the tab **Search** then type "my Author" in the seach text box, you will see the following result:

Note: The source files can be found in the zip file attached to this page, see the end of this page

6.3. WebDav Access

Since version 2.2, ECM changed the behavior to access WebDAV server.

We split to two ways:

Public mode:

http//host:port-rest-jcr-repositoryName}/workspaceName}/nodePath}

Private mode:

http//host:port-rest-private-jcr-repositoryName}/workspaceName}/nodePath}

The difference between Public mode and Private mode is: When accessing the WebDAV server, if the URL is Private mode, WebDAV will ask users to enter username and password. And if the URL is Public mode, there is no need to enter username and password, of course see only the public nodes shared with everyone.

You can see changes in the web.xml file at the location exo.ecm.web.ecmportal/src/main/webapp/WEB-INF/

Change param-value for this filter

```
<filter>
  <filter-name>AnonymousUserContextRedirectionFilter</filter-name>
  <filter-class>org.exoplatform.ws.frameworks.servlet.AnonymousUserContextRedirectionFilter</filter-class>
              <init-param>
    <param-name>context-name</param-name>
    <param-value>/rest/private</param-value>
  </init-param>
</filter>
```

and filter mapping also

```
<filter-mapping>
  <filter-name>RestEncodingFilter</filter-name>
  <url-pattern>/rest/*</url-pattern>
</filter-mapping>

<filter-mapping>
<!-- This must be before ThreadLocalSessionProviderInitializedFilter for URI /rest/private/* -->
  <filter-name>AnonymousUserContextRedirectionFilter</filter-name>
  <url-pattern>/rest/private/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>ThreadLocalSessionProviderInitializedFilter</filter-name>
  <url-pattern>/rest/private/*</url-pattern>
</filter-mapping>
```

See more details in the following URL: http://wiki.exoplatform.org/xwiki/bin/view/JCR/WebDav But note that, the difference is the URL

6.4. User-Defined Document Types

6.5. Create a Node Type

Do following steps to create a new node type :

1 Go to **ECM Administration** -> **Type of Content** -> **Manage Node Type**.

2. Click the **Add** button to show the form to create a new node type.

3. Select a name space for the node. (A namespace is like a prefix of a node).

4. Enter the name for the node in 'Node type name'. This field is required.

5. Select a value for the **Is mixin type** field.

- True: this node is mixin type.
- False:this node is not mixin type.

6. Select a value for the **Orderable child nodes** field:

- True: child nodes are ordered.

- False:child node are not ordered.

7. Enter a value for the **Primary item name** field.

8. **Super types**: click to add more parent node.

9. Property definitions: list all definition names of the **Property** tab.

10. Child node definitions: list all definition names of the **Child node** tab.

11 Click the **Save** button to accept adding a new node type or click the **Save draft** button to save this node type as draft

6.6. Create a Template

Each created node needs to have a form to enter data for their properties (called dialog template) and to display the existing values (called view template). Following guides to create a new template:

1 Go to **ECM Administration** -> **Content Presentation** -> **Manage Templates**.

2. Click the **Add** button on the **Manage Templates** form to open the **Add new template** form.

3. Select the type of the template that you want to add by clicking in the **Name** field.

4. Add a label for this template in the **Label** field.

5. Check /uncheck **Document Template** if this template is for document or not.

6. Select permissions for this template.

7. Select the **Dialog** tab and enter a value into the **Dialog content** field.

8. Select the **View** tab and enter values into the **View content** field.

9. Click the **Save** button to accept adding a new template.

6.7. Use the JCR FE to create the Document

After creating new document templates, users will use in File Explore, do as follows:

1 Go to **ECM Administration** -> **File Explorer**.

2. Select any drive.

3. Select the **Action** tab and click the **Add Document** icon (or right click on the right pane and select 'Add Document') to open a form to add a document.

4. Select the type of document that you have just created in ECM Administration.

5. Put values in fields.

6. Click the **Save** button.

<h3> **How to do create new node type with one text property and one date property, using XML**

configuration files ? </h3> (edit in webapps/portal/WEB-INF/conf/dms/nodetypes-ecm.xml file)

```
<nodeType name="exo:newnodetype" isMixin="true" hasOrderableChildNodes="false" primaryItemName="">
    <supertypes>
      <supertype>exo:article</supertype>
    </supertypes>
    <propertyDefinitions>

      <propertyDefinition name="text" requiredType="String" autoCreated="true" mandatory="true" onParentVersion=
        protected="false" multiple="false">
        <valueConstraints/>
      </propertyDefinition>
      <propertyDefinition name="date" requiredType="Date" autoCreated="false" mandatory="true" onParentVersion="
        protected="false" multiple="false">
        <valueConstraints/>
      </propertyDefinition>

    </propertyDefinitions>
  </nodeType>
```


 <h3> **How to create new template using XML configuration files ?** </h3> (edit in webapps/portal/WEB-INF/conf/dms/dms-templates-configuration.xml file)

```
<value>
    <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType">
       <field  name="nodetypeName"><string>exo:newnodetype</string></field>
       <field  name="documentTemplate"><boolean>true</boolean></field>


       <field  name="label"><string>AAAAA</string></field>
       <field  name="referencedView">
          <collection type="java.util.ArrayList">
             <value>
               <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                   <field name="templateFile"><string>/fornewnodetype/views/view1.gtmpl</string>
                    </field>
                   <field name="roles"><string>*</string></field>
               </object>
             </value>
          </collection>
       </field>
       <field  name="referencedDialog">
          <collection type="java.util.ArrayList">
             <value>
                 <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                    <field name="templateFile"><string>/fornewnodetype/dialogs/dialog1.gtmpl</string>
                    </field>
                    <field name="roles"><string>*</string></field>
                 </object>
               </value>
           </collection>
       </field>
     </object>
  </value>
```

6.8. Direct Access of JCR Drives

When you access the **File Explorer** normally, it shows several drives. The use sees only those drives she or he has access to. The user can then choose the drive he wishes to work with.

By configuring the portlet settings you can modify this default behavior. By default the "selection" option is configured. Besides "selection", there are 4 other ways to configure the file explorer :

- Jailed
- Personal
- Social
- Parameterize

Context adaptation of eXo JCR File Explorer

In eXo DMS (Document Management System), users manage the content through an application called "JCR File Explorer". This application allows creating, retrieving, updating or deleting content. By default, it forces users to initially select a drive. A drive can be considered as a logical unit of storage (for example: live documents, user private documents, workgroup documents).

The selection step is not really convenient and in order to improve productivity JCR File Explorer can optionally bypass the drive selection step and let the users directly access the drive matching the context of their work.

Of course, in all cases, security checks are performed to ensure the current user is allowed to access the target location.

6.8.1. Configuration by the User Interface

- Move the mouse to the right corner of this portlet and select the **Edit** mode. A form appears to edit File Explorer

- Click the **Edit** button

From here, the User Interface depends on the mode you chose, so please directly refer to the specific mode parts below:

6.8.1.1. Selection

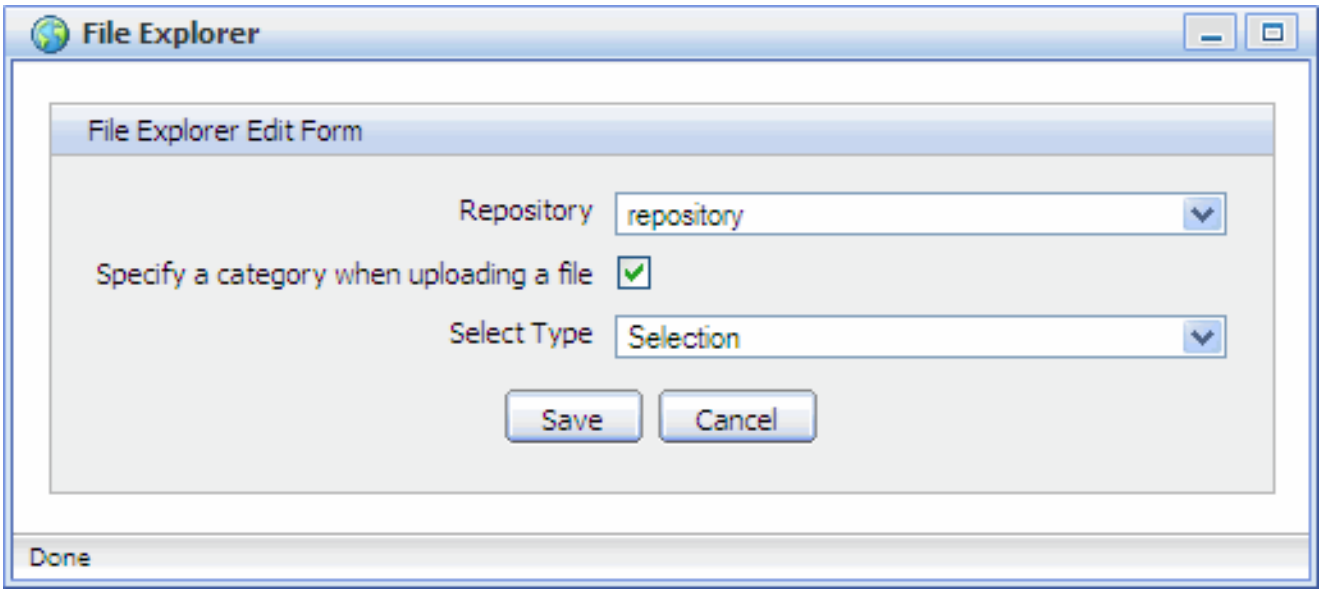


Table 6.1.

Field	Meaning
Repository	Select the repository which contains the drive that a user wants to select
Specify a category when uploading a file	Force an user to add a category when uploading or creating a document

6.8.1.2. Jailed

In the "jailed" use case, the administrator configures a unique drive. Only that drive is accessible by the file explorer instance (http://en.wikipedia.org/wiki/Jail_%28computer_security%29). In Unix system the same mechanism is achieved by using "chroot".

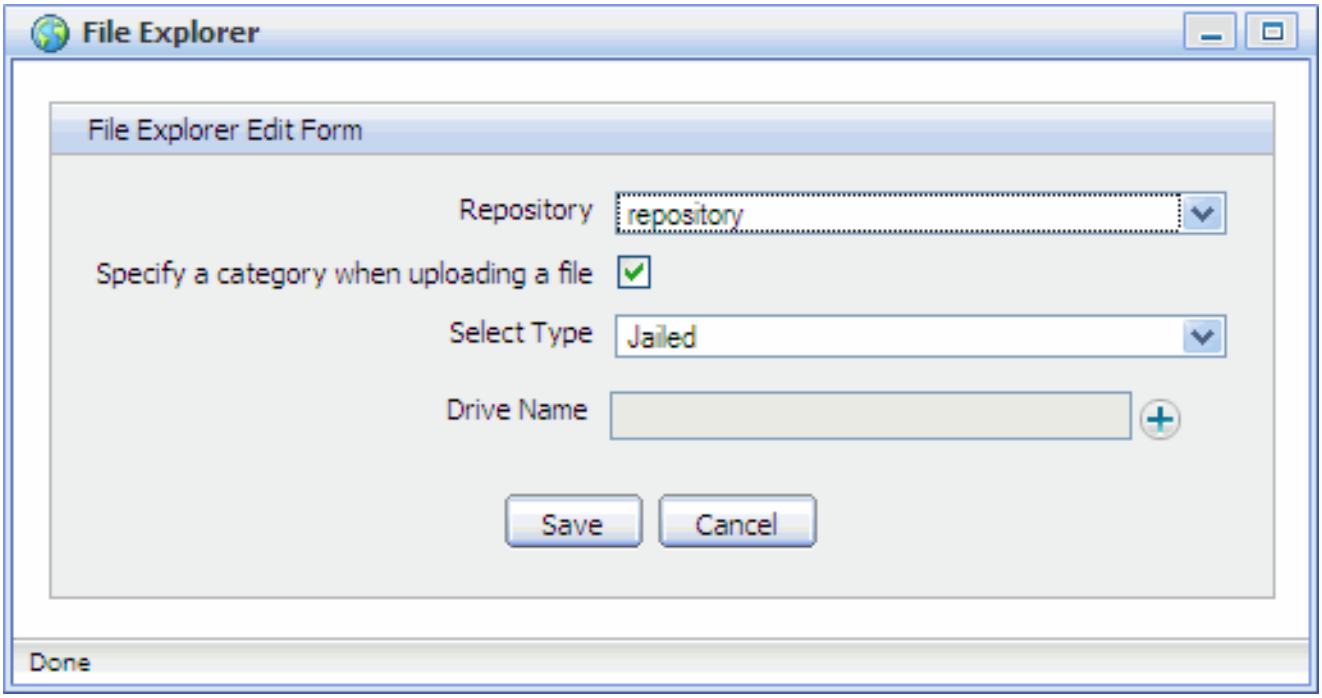
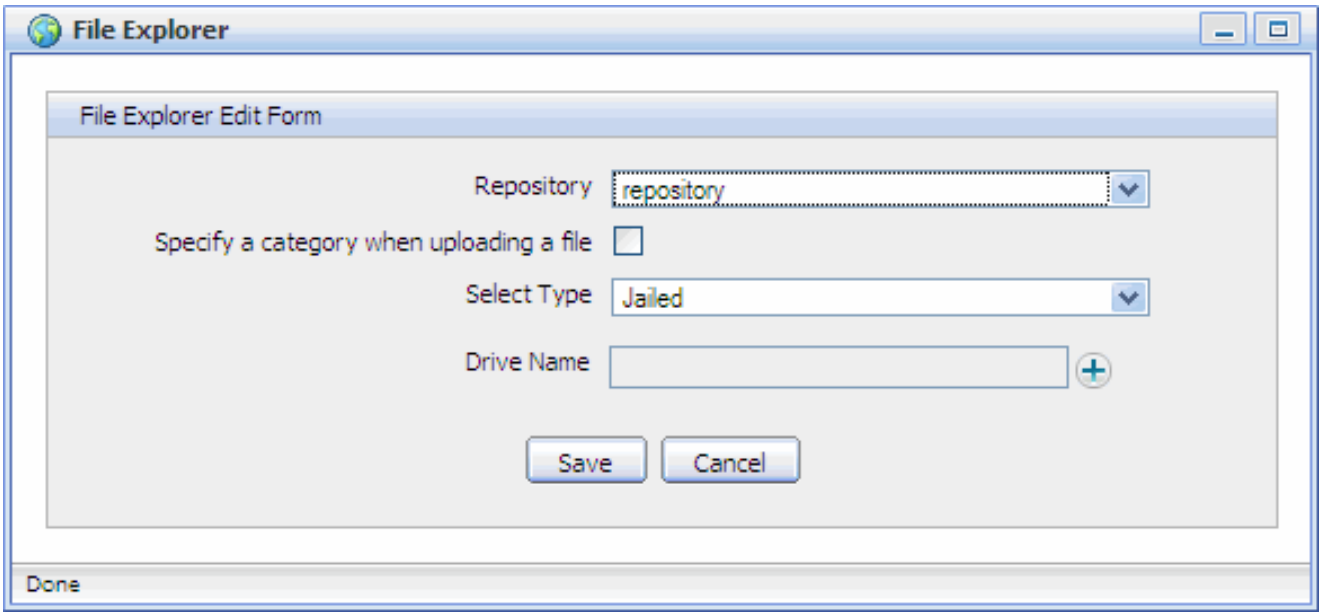
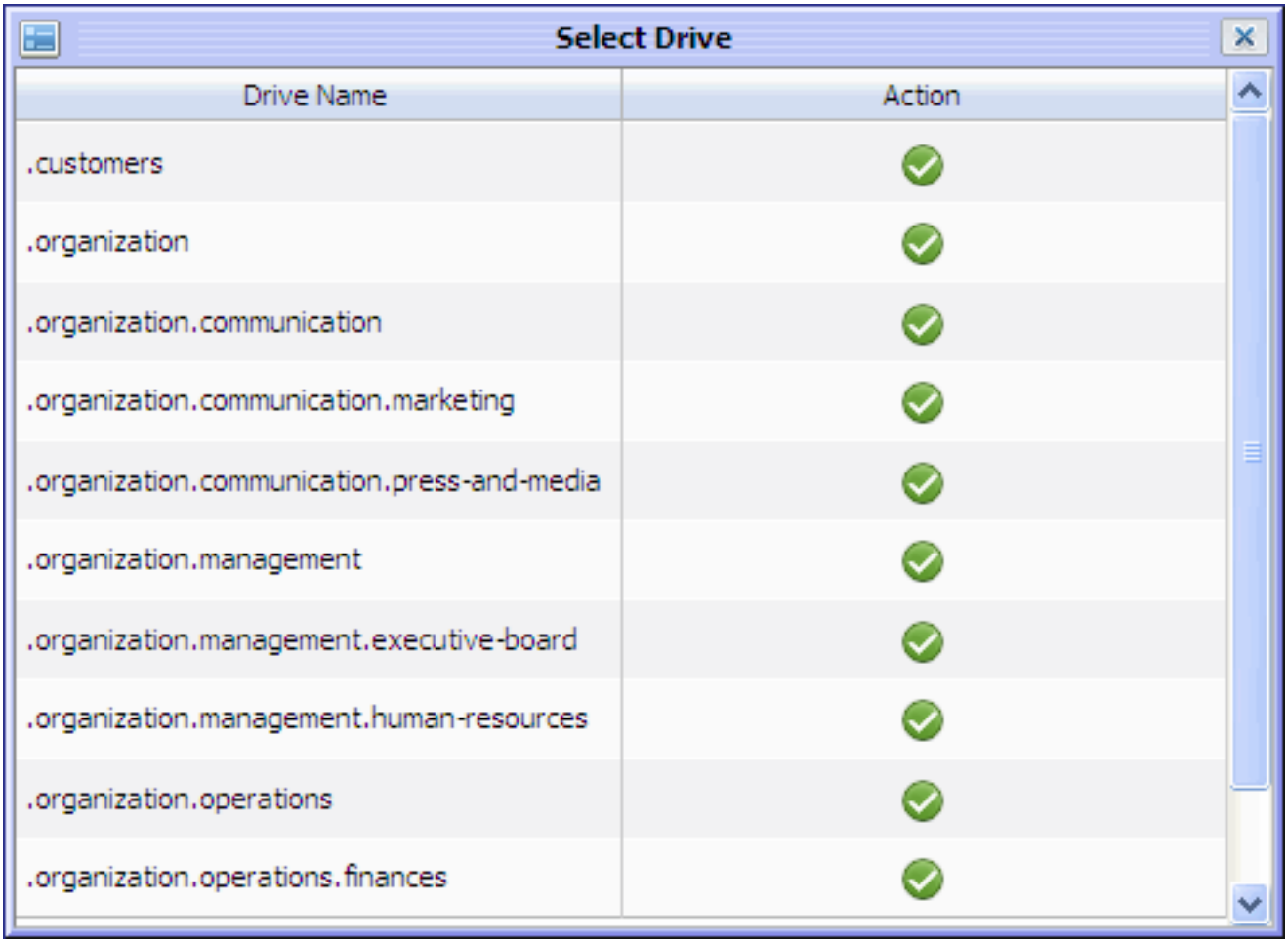


Table 6.2.

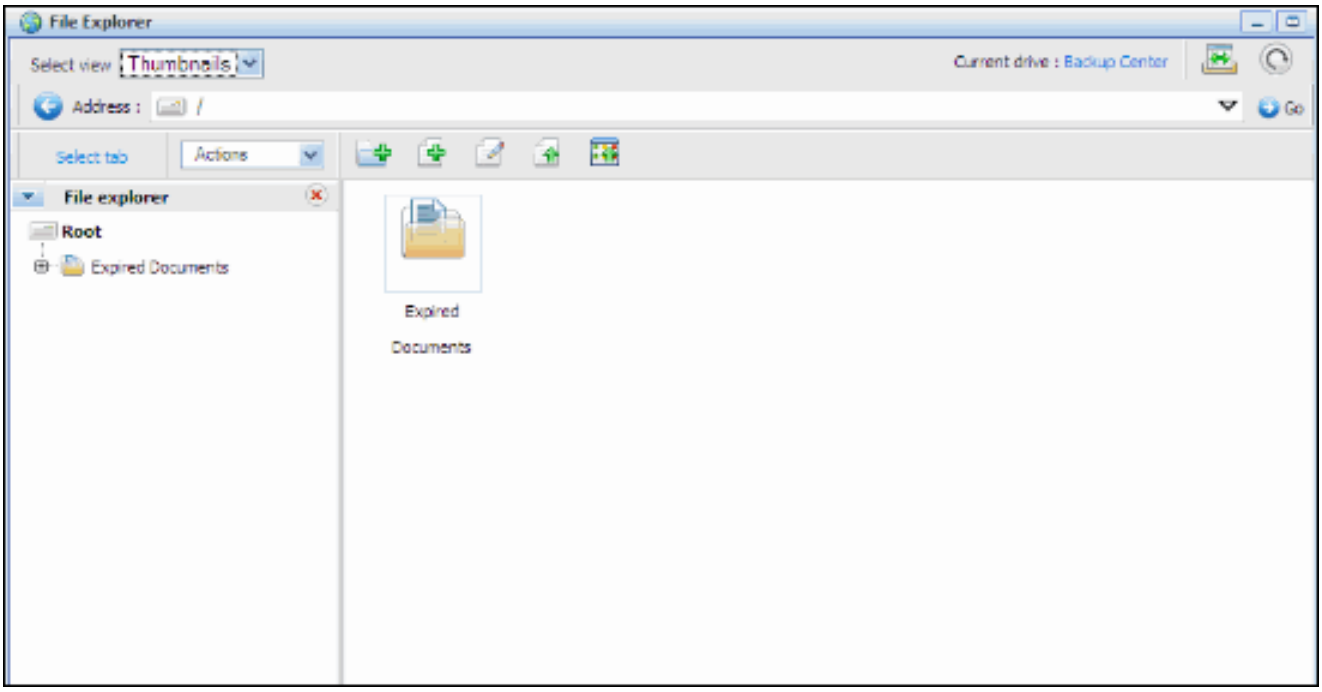
Field	Meaning
Repository	Select the repository which contains the drive that an user wants to select
Specify a category when uploading a file	Force an user to add a category when uploading or creating a document
Drive Name	The name of a drive that an user wants to access

You can select a drive by clicking on  . The following drive will appear:





As you can see in the picture below, the 'Back to JCR main Drives' icon doesn't appear anymore



6.8.1.3. Parameterized

TODO: Mention the URL parameters !

In the "parameterized" use case, the application fetches some location information from the portal page URL. Then it will directly access that specified location in the storage (can be a folder or a document). This use case is handy to allow linking the JCR File Explorer from external applications. A typical example is a Google gadget listing the last 5 modified documents. When clicking, the user is redirected to eXo Portal and directly accesses the document in its context.

6.8.1.4. Personal

When you choose the Personal usecase, the Private drive of the current user will be used.

Table 6.3.

Field	Meaning
Repository	Select the repository which contains the drive that an user wants to select
Specify a category when uploading a file	Force an user to add a category when uploading or creating a document

6.8.1.5. Social

In the "social" use case, the JCR File Explorer will directly access the drive containing content shared by the community he's connected to (this use case is compatible with the eXo Spaces module).

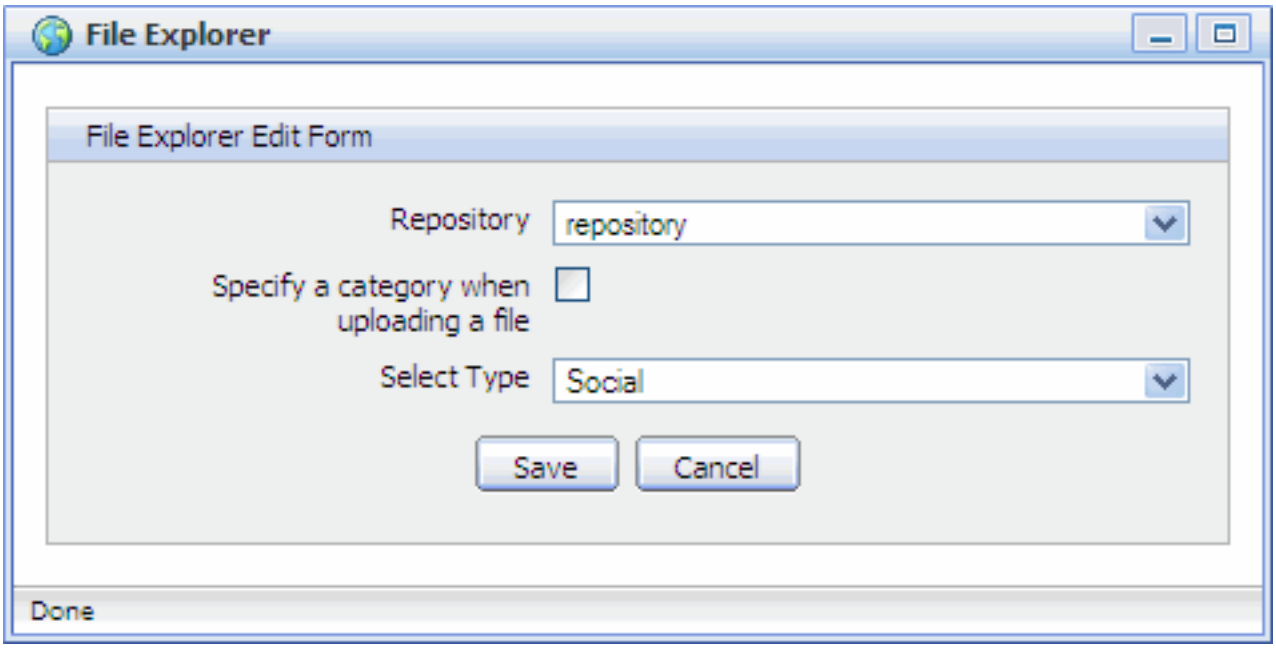


Table 6.4.

Field	Meaning
Repository	Select the repository which contains the drive that an user wants to select
Specify a category when uploading a file	Force an user to add a category when uploading or creating a document

6.8.2. Configuration by xml files

In the **portlet.xml** and **portal/group/platform/user/portlet-preferences.xml**, insert new preference values

```
<preference>
 <name>usecase</name>
 <value>selection</value>
 <read-only>false</read-only>
</preference>
<preference>
 <name>driveName</name>
 <value>Collaboration Center</value>
 <read-only>false</read-only>
</preference>
```

Table 6.5.

Field	Default value	Explain
usecase	selection	The ways to access the drive.
driveName	Collaboration Center	The parameter is specified to decide which the driveName is.

6.9. Forcing Taxonomies at File Upload

Note

Since ECM 2.2

Since ECM 2.2, in the File Explorer portlet, you can add references to the file which will be uploaded. By default this function is not mandatory. You can change the value equal true in the parameter **categoryMandatoryWhenFileUpload** to make sure every files will be added the categories when uploaded. You can see and change the configuration if you want in the file exo.ecm.portlet.ecm/src/main/webapp/WEB-INF/portlet.xml

```
<preference>
  <name>categoryMandatoryWhenFileUpload</name>
  <value>false</value>
  <read-only>false</read-only>
</preference>
```


6.10. View Service Configuration

Manage view service allows administrators to customize many views that can affect users when exploring workspaces. Permissions, template and actions tabs are the view foundations.

```
  <component>
    <key>org.exoplatform.services.cms.views.ManageViewService</key>
    <type>org.exoplatform.services.cms.views.impl.ManageViewServiceImpl</type>
    .......
  </component>
```

The service can handle plugins

```
    <component-plugins>
      <component-plugin>
        <name>manage.view.plugin</name>
        <set-method>setManageViewPlugin</set-method>
        <type>org.exoplatform.services.cms.views.impl.ManageViewPlugin</type>
        <description>this plugin manage user view</description>
        .........
      </component-plugin>
    </component-plugins>
```

The plugin parameters are now explained

```
    <init-params>
      <object-param>
        ....
      </object-param>
      ......
    </init-params>
```

Parameters are used to create default views, templates and actions of Manage view service. Each object-param

have a type that is a class representing all properties of a view.

View data object parameters: There are three predefined views, admin-view(view for users in admin group, that view let users have full actions when exploring workspaces, default template of this view is list view), simple-view(view for users in users group, users in this group have limited actions, default template of this view is thumbnails view), anonymous-view(view for quest user, they only have search action with thumbnails view template).

```
<object-param>
  <name>Admin-View</name>
  <description>View configuration of Admin</description>
  <object type="org.exoplatform.services.cms.views.ViewConfig">
    <field  name="name"><string>admin-view</string></field>
    <field  name="permissions"><string>member:/admin</string></field>
    <field  name="template"><string>/jcr:system/exo:ecm/views/templates/ecm-explorer/ListView</string></fiel
    <field  name="tabList">
      <collection type="java.util.ArrayList">
        <value>
          <object type="org.exoplatform.services.cms.views.ViewConfig$Tab">
            <field  name="tabName"><string>Admin</string></field>
            <field  name="buttons">
              <string>
                manageVersions; manageCategories; manageRelations; manageActions; exportNode; importNode
              </string>
            </field>
          </object>
        </value>
        <value>
          <object type="org.exoplatform.services.cms.views.ViewConfig$Tab">
            <field  name="tabName"><string>Info</string></field>
            <field  name="buttons">
              <string>
                viewReferences; viewNodeType; viewPermissions; viewProperties; viewRelations
              </string>
            </field>
          </object>
        </value>
        <value>
          <object type="org.exoplatform.services.cms.views.ViewConfig$Tab">
            <field  name="tabName"><string>Actions</string></field>
            <field  name="buttons">
              <string>
                addFolder; addDocument; upload; createTicket; paste; search
              </string>
            </field>
          </object>
        </value>
      </collection>
    </field>
  </object>
</object-param>
```

If property type is a list of object also config value of property as collection of specify object type

```
  <field  name="tabList">
    <collection type="java.util.ArrayList">
      <value>
        <object type="org.exoplatform.services.cms.views.ViewConfig$Tab">
          <field  name="tabName"><string>Admin</string></field>
          <field  name="buttons">
            <string>
              manageVersions; manageCategories; manageRelations; manageActions; exportNode; importNode
            </string>
          </field>
        </object>
      </value>
      .......
    </field>
  </collection>
```

Template data object parameter:

```
    <object-param>
      <name>List Template</name>
      <description>Template for display documents in list style</description>
      <object type="org.exoplatform.services.cms.views.TemplateConfig">
        <field  name="type"><string>ecmExplorerTemplate</string></field>
        <field  name="name"><string>ListView</string></field>
        <field  name="warPath"><string>/ecm-explorer/ListView.gtmpl</string></field>
      </object>
    </object-param>
    ..........
```

Value Parameter: Configure all actions in views

```
    <init-params>
      <value-param>
        <name>buttons</name>
        <value>
          addFolder; addDocument; upload; createTicket; paste; search; viewReferences; viewNodeType;
          viewPermissions; viewProperties; manageVersions; manageCategories; manageRelations;
          manageActions; exportNode; importNode; viewRelations
        </value>
      </value-param>
    </init-params>
```

6.11. Edit and View Mode Configuration

6.11.1. With the Edit mode

In the portlet.xml file

```
<portlet>
    <description xml:lang="EN">Fast Content Creator</description>
    <portlet-name>FastContentCreatorPortlet</portlet-name>
    <display-name xml:lang="EN">Fast Content Creator</display-name>
    <portlet-class>org.exoplatform.webui.application.portlet.PortletApplicationController</portlet-class>

    <init-param>
      <name>webui.configuration</name>
      <value>/WEB-INF/conf/portlet/ecm/FastContentCreatorPortlet/webui/configuration.xml</value>
    </init-param>

    <expiration-cache>0</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>help</portlet-mode>
      <portlet-mode>edit</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
    <resource-bundle>locale.portlet.ecm.FastContentCreatorPortlet</resource-bundle>

    <portlet-info>
      <title>Fast Content Creator</title>
      <short-title>ECM Fast Content Creator</short-title>
      <keywords>ecm, fastcontentcreator</keywords>
    </portlet-info>
    <portlet-preferences>
   <!-- Setup the default repository name for fast content creator portlet -->
     <preference>
        <name>repository</name>
        <value>repository</value>
        <read-only>false</read-only>
      </preference>
      <!-- Specify the workspace name will be use in repository -->
```

```
      <preference>
        <name>workspace</name>
        <value>collaboration</value>
        <read-only>false</read-only>
      </preference>
      <!-- Specify the destination path which will be use to store saved documents -->
      <preference>
        <name>path</name>
        <value>/Groups/platform/users/Documents</value>
        <read-only>false</read-only>
      </preference>
      <!-- Specify the node type of document which will be show on the dialog form -->
      <preference>
        <name>type</name>
        <value>exo:article</value>
        <read-only>false</read-only>
      </preference>
    </portlet-preferences>
  </portlet>
```


 In configuration.xml file

```
<webui-configuration>
  <annotation-classes>
  </annotation-classes>
  <application>
    <ui-component-root>org.exoplatform.ecm.webui.component.fastcontentcreator.UIFastContentCreatorPortlet</ui-co
    <state-manager>org.exoplatform.webui.application.portlet.ParentAppStateManager</state-manager>
  </application>
</webui-configuration>
```


6.11.2. With the View mode

In portlet-preferences.xml file

```
<portlet-preferences>
    <owner-type>group</owner-type>
    <owner-id>platform/users</owner-id>
    <window-id>group#platform/users:/ecm/FastContentCreatorPortlet/fastcontentcreator</window-id>
    <preference>
        <name>repository</name>
        <value>repository</value>
        <read-only>false</read-only>
      </preference>
      <preference>
        <name>workspace</name>
        <value>collaboration</value>
        <read-only>false</read-only>
      </preference>
      <preference>
        <name>path</name>
        <value>/Groups/platform/users/Documents</value>
        <read-only>false</read-only>
      </preference>
      <preference>
        <name>type</name>
        <value>exo:article</value>
        <read-only>false</read-only>
      </preference>
   </portlet-preferences>
```


 In pages.xml file

```
<page>
```

```
   <page-id>group::platform/users::FastContentCreator</page-id>
   <owner-type>group</owner-type>
    <owner-id>platform/users</owner-id>
    <name>FastContentCreator</name>
    <access-permissions>*:/platform/users</access-permissions>
    <edit-permission>manager:/platform/administrators</edit-permission>

    <application>
      <instance-id>group#platform/users:/ecm/FastContentCreatorPortlet/fastcontentcreator</instance-id>
      <title>Fast Content Creator</title>
      <show-info-bar>true</show-info-bar>
      <show-application-mode>true</show-application-mode>
    </application>
  </page>
```


 In navigation.xml file

```
<page-nodes>
   <node>
<node>
      <uri>contentmanagement/fastcontentcreator</uri>
      <name>fastcontentcreator</name>
      <label>#{platform.users.content-creator}</label>
      <page-reference>group::platform/users::FastContentCreator</page-reference>
    </node>
</node>
  </page-nodes>
```

6.12. Predefined Taxonomies

6.13. Service Definition

The service which defines categories is configured into the jar *exo.ecm.dms.component.cms-XXX.jar* (*exo.ecm.component.cms-XXX.jar* in old versions) as follow:

```
<component>
  <key>org.exoplatform.services.cms.categories.CategoriesService</key>
  <type>org.exoplatform.services.cms.categories.impl.CategoriesServiceImpl</type>
  ..............
</component>
```

1 Plugin Definition

The service accepts plugins that can define new taxonomies.

1.1 Component Plugin Part

When the application will process the code below, it will call the method addTaxonomyPlugin(org.exoplatform.services.cms.categories.impl.TaxonomyPlugin) on the implementation of the service org.exoplatform.services.cms.categories.CategoriesService.

```
<external-component-plugins>
  <target-component>org.exoplatform.services.cms.categories.CategoriesService</target-component>
  <component-plugin>
    <name>predefinedTaxonomyPlugin</name>
    <set-method>addTaxonomyPlugin</set-method>
    <type>org.exoplatform.services.cms.categories.impl.TaxonomyPlugin</type>
    <init-params>
    ..............<!-- This part is detailed later in this article -->
```

```
    </init-params>
  </component-plugin>
</external-component-plugins>
```

6.13.1. Initial Parameters Part

To be created the class *org.exoplatform.services.cms.categories.impl.TaxonomyPlugin* needs the following initial parameters

```
<init-params>
  <value-param>
    <name>autoCreateInNewRepository</name>
    <value>true</value>
  </value-param>
  <value-param>
    <name>repository</name>
    <value>repository</value>
  </value-param>
  <object-param>
    <name>taxonomy.configuration</name>
    <description>configuration predefined taxonomies to inject in jcr</description>
    <object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig">
     ..............  <!- This part is detailed later in this article ->
    </object>
  </object-param>
</init-params>
```

Table 6.6.

Property Name	**Purpose**
autoCreateInNewRepository	Indicates if the taxonomies must be created in all the repositories available
repository	The name of the repository in which we want to create the taxonomies. Only used if the parameter "autoCreateInNewRepository" has been set to false.
taxonomy.configuration	The object parameter in which we define the taxonomies to create.

1.1 Taxonomy Configuration Part

In this part we define the list of all the categories/taxonomies we would like to create automatically. The example below create the taxonomy called "**myTaxonomyName**" which corresponding path is "**/my/taxonomy/absolute/path**".

```
<object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig">
  <field name="taxonomies">
  <collection type="java.util.ArrayList">
    <value>
    <object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig$Taxonomy">
      <field name="name"><string>myTaxonomyName</string></field>
      <field name="path"><string>/my/taxonomy/absolute/path</string></field>
    </object>
    </value>
    ..............
  </collection>
  </field>
</object>
```

Table 6.7.

Property Name	Purpose
name	The name of the taxonomy
path	The absolute path of the taxonomy

6.14. Default Configuration Location

The configuration file which defines all the default taxonomies, is located at */portal/WEB-INF/conf/dms/dms-categories-configuration.xml* (*/portal/WEB-INF/conf/ecm/ecm-categories-configuration.xml* in old versions)

6.15. How is it working ?

The taxonomy is stored in the JCR at the path */jcr:system/exo:ecm/exo:taxonomies_*. *All nodes of the taxonomy have the type* exo:taxonomy_. The name of the category is the path to this category so for example */cms/sports/football/champions-league*.

1. categorized documents have the mixin *exo:categorized_ and in the property* exo:category_ that contains the list of references to the taxonomy tree.

6.16. Notes

Note

You can only predefine taxonomies when the Java Content Repository has never been initialized. Otherwise, you need to create them manually from the User Interface

6.17. Audit Trails Configuration

6.18. Introduction

Audit Trails in eXo are based on JCR operations. The corresponding interface is described in Audit Trails Management.

6.19. Configuration File

You find the audit configuration file here: ...portal/WEB-INF/conf/dms/jcr-component-plugins-configuration.xml

The current default configuration looks like:

```
  <external-component-plugins>
    <target-component>org.exoplatform.services.jcr.impl.ext.action.SessionActionCatalog</target-component>

<component-plugin>
            <name>addActions</name>
            <set-method>addPlugin</set-method>
            <type>org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin</type>
            <description>add actions plugin</description>
            <init-params>
              <object-param>
                <name>actions</name>
                <object type="org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin$ActionsConfig">
                  <field  name="actions">
                    <collection type="java.util.ArrayList">
<!-- Uncomment this tag if you wish to enable audit in the entire workspace by default.
                  <value>
                    <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
                      <field  name="eventTypes"><string>addNode</string></field>
                      <field name="path"><string>/</string></field>
                      <field  name="isDeep"><boolean>true</boolean></field>
                      <field name="actionClassName"><string>org.exoplatform.services.jcr.ext.audit.AddAuditableA
                   </object>
                  </value>
-->
                      <value>
                        <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
                          <field  name="eventTypes"><string>addProperty,changeProperty,removeProperty</string></
                          <field  name="path"><string>/</string></field>
                          <field  name="nodeTypes"><string>exo:auditable</string></field>
                          <field  name="isDeep"><boolean>true</boolean></field>
                          <field  name="actionClassName"><string>org.exoplatform.services.jcr.ext.audit.AuditAct
                        </object>
                      </value>
                      <value>
                        <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
                          <field  name="eventTypes"><string>addMixin</string></field>
                          <field  name="path"><string>/</string></field>
                          <field  name="nodeTypes"><string>exo:auditable</string></field>
                          <field  name="isDeep"><boolean>false</boolean></field>
                          <field  name="actionClassName"><string>org.exoplatform.services.jcr.ext.audit.AuditAct
                        </object>
                      </value>
                      <value>
                        <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
                          <field  name="eventTypes"><string>removeNode</string></field>
                          <field  name="nodeTypes"><string>exo:auditable</string></field>
                          <field  name="isDeep"><boolean>false</boolean></field>
                          <field  name="actionClassName"><string>org.exoplatform.services.jcr.ext.audit.RemoveAu
                        </object>
                      </value>
                    </collection>
                  </field>
                </object>
              </object-param>
            </init-params>
          </component-plugin>
  </external-component-plugins>
```

1 Details

Be aware of this simple rule: An audit trail is created only if all conditions you have configured are fulfilled. If

- the concerned event corresponds with of the event types,
- the concerned node has one of the node types (if you configured a node type),
- the concerned node matches one of the workspaces you configured (if you configured one), and

- the concerned node matches one of the paths you configured - or any descendant if you configured "true" for "isDeep" (if you configured a path),

then the action you configured in "actionClassName" is triggered.

1.1 Event Types

There are several event types whose names are self-explaining:

- addNode
- addProperty
- changeProperty
- addMixin
- removeProperty
- removeNode
- removeMixin

1.1 Path and Deepness In the default configuration you see that the path is always "/". That means the root node of the workspaces you configured. If "isDeep" is true all descendants of the path matches the path condition. If "isDeep" is false only the path and its child nodes matches the path condition. "/" together with "true" means the whole workspace matches the path condition.

If you wish to enter several paths you have to separate them by commas.

1.1 Auditable NodeType Always use the exo:auditable node type.

If you don't want to have a node type condition., you just don't provide any node type as you see in the part that is commented in the above configuration.

1.1 Workspace In the example you don't see any information about the workspace. In that case all workspaces are part of the configuration. To restrict the configuration to certain workspaces you must add for example:

```
<field  name="workspace"><string>collaboration,system</string></field>
```

To configure for the workspaces "collaboration" and "system".

Chapter 7. Integration

7.1. Integrating Content Validation into DMS

7.2. 1.Current situation

We now need to propose two products to our customers:

-
  - DMS. It contains a set of Portlets and services aimed at managing documents in the JCR.
  - Workflow. It contains a set of Portlets and services aimed at managing and interfacing with business processes engines. These products are not coupled and have to be generated separately.

Optionally, we have to provide content validation option on top of DMS. We cannot do that by default, for royalties reasons. When this option is turned on, we need to merge Workflow into DMS and also add a set of services and business processes to the packaging.

In SVN, this was implemented the following way:

```
ecm
|___dms/
|___workflow/
|___content-validation/
```

content-validation was detached for two reasons:

-
  - it looked easier to manage separately permissions for differentiating customers who bought a subscription for DMS and those who bought the content-validation option.
  - if we put content-validation inside DMS package, then we thought we had to create a link from DMS to Workflow, which is not wished.

7.3. 2.The problem

We now have a content-validation entry in SVN, which has "trunk", "tags" and "branches". This means it should be considered as a plain project, with a dedicated lifecycle and a JIRA project. The granularity is not well appropriated for that.

7.4. 3.The solution

We are integrate content-validation inside DMS, which sounds more consistent by using Maven profile:

```
ecm
|___dms/
|     |___trunk/
|              |___ core/[Current DMS code goes here]
```

```
|              |___ ext/content-validation/
|___workflow/
```

So, in DMS we edit **dms/trunk/pom.xml** to build **dms/trunk/core** by default:

```
<project>

  <parent>
    <groupId>org.exoplatform</groupId>
    <artifactId>parent</artifactId>
    <version>1.0.0</version>
  </parent>

  <modelVersion>4.0.0</modelVersion>
  <groupId>org.exoplatform.ecm</groupId>
  <artifactId>dms</artifactId>
  <name>eXo DMS Config</name>
  <version>2.3-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>core</module>
  </modules>
  ...
```

And for using maven profile to build content validation, we declare in this **pom.xml** file:

```
<profiles>
    <profile>
      <id>default</id>
      <activation>
        <property><name>default</name></property>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <org.exoplatform.kernel.version>2.0.6</org.exoplatform.kernel.version>
        <org.exoplatform.core.version>2.1.4</org.exoplatform.core.version>
        <org.exoplatform.jcr.version>1.10.2</org.exoplatform.jcr.version>
        <org.exoplatform.ws.version>1.3.2</org.exoplatform.ws.version>
        <org.exoplatform.pc.version>2.0.4</org.exoplatform.pc.version>
        <org.exoplatform.portal.version>2.5.2</org.exoplatform.portal.version>
        <org.exoplatform.ecm.dms.version>2.3-SNAPSHOT</org.exoplatform.ecm.dms.version>
                <exo.test.classes>Test</exo.test.classes>
        <exo.test.skip>true</exo.test.skip>
        <bonita.version>3.0</bonita.version>
      </properties>
    </profile>
        <profile>
            <id>ext</id>
            <activation>
              <property>
                <name>ext</name>
              </property>
              <activeByDefault>false</activeByDefault>
            </activation>
            <modules>
              <module>/ext</module>
            </modules>
    </profile>
  </profiles>
```

In **dms/trunk/core**, we create **pom.xml** file to build all module of current DMS

```
<project>
  <parent>
    <groupId>org.exoplatform.ecm.dms</groupId>
    <artifactId>config</artifactId>
    <version>2.3-SNAPSHOT</version>
  </parent>
```

```
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.exoplatform.ecm.dms.core</groupId>
  <artifactId>config</artifactId>
  <name>eXo DMS Config</name>
  <version>2.3-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>component/cms</module>
        <module>component/deployment</module>
        <module>component/publication</module>
        <module>connectors/fckeditor</module>
        <module>webui/dms</module>
        <module>web/eXoDMSResources</module>
  <module>portlet/ecm</module>
  <module>portlet/jcr-console</module>
    <module>web/dmsportal</module>
  </modules>

  <build>
    <resources>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
          <include>**/*.gtmpl</include>
          <include>**/*.gif</include>
        </includes>
      </resource>
    </resources>
    <testResources>
      <testResource>
        <directory>src/test/java</directory>
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
        </includes>
      </testResource>
    </testResources>
   ....

  <profiles>
    <profile>
      <id>default</id>
      <activation>
        <property><name>default</name></property>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <org.exoplatform.kernel.version>2.0.6</org.exoplatform.kernel.version>
        <org.exoplatform.core.version>2.1.4</org.exoplatform.core.version>
        <org.exoplatform.jcr.version>1.10.2</org.exoplatform.jcr.version>
        <org.exoplatform.ws.version>1.3.2</org.exoplatform.ws.version>
        <org.exoplatform.pc.version>2.0.4</org.exoplatform.pc.version>
        <org.exoplatform.portal.version>2.5.2</org.exoplatform.portal.version>
        <org.exoplatform.ecm.dms.version>2.3-SNAPSHOT</org.exoplatform.ecm.dms.version>
        <exo.test.classes>Test</exo.test.classes>
        <exo.test.skip>true</exo.test.skip>
        <bonita.version>3.0</bonita.version>
      </properties>
    </profile>
  </profiles>
</project>
```

And create **pom.xml** file in **ecm/dms/trunk/ext/** to build **contenvalidation** (and other module if have)

```
<project>

  <parent>
    <groupId>org.exoplatform.ecm.dms</groupId>
    <artifactId>config</artifactId>
    <version>2.3-SNAPSHOT</version>
  </parent>
```

```
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.exoplatform.ecm.dms</groupId>
  <artifactId>ext</artifactId>
  <name>eXo Content Validation</name>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>contentvalidation</module>
  </modules>

  <profiles>
    <profile>
      <id>default</id>
      <activation>
        <property><name>default</name></property>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <exo.test.skip>true</exo.test.skip>
      </properties>
    </profile>
  </profiles>
</project>
```

7.5. 4. Build DMS

4.1 Build DMS with core: by default in **dms/trunk/** we run:

```
mvn clean install
```

4.2 Build DMS with **dms/core** and **dms/contentvalidation**, we use maven profile in **dms/trunk/**, run

```
mvn clean install -P ext
```

4.3 Build and deploy **dms** version **trunk**

```
exobuild --product=dms --exclude=all --build --deploy
```

4.4 Build and deploy **dms** version **trunk** with **workflow**

```
exobuild --product=dms --exclude=all --build --deploy --enable-workflow
```

Because from now, we change in **tools/trunk/build/src/main/javascript/eXo/command/exobuild.js** to use maven profile:

```
  if (enableWorkflow && (product.name == "eXoDMS")) {
    mvnArgs = ["clean", "install", "-P", "ext"];
  } else {
    mvnArgs = ["clean", "install"];
  }
```

Chapter 8. Migration

8.1. ECM Migration from 2-0-x to 2-1

8.2. Migrate your portal to be compatible with portal 2.2

Click here for more details

> **Note**
>
> This step is important: auto-init-root-nodetype and auto-init-permissions are DEPRECATED in JCR 1.9

8.3. Add a new parameter for the AuditService

In the war:/conf/jcr/jcr-configuration.xml add a new parameter to the org.exoplatform.services.jcr.ext.audit.AuditService as follow:

```
<component>
  <key>org.exoplatform.services.jcr.ext.audit.AuditService</key>
  <type>org.exoplatform.services.jcr.ext.audit.AuditServiceImpl</type>
  <init-params>
    ...
    <value-param>
      <name>adminIdentity</name>
      <value>*:/platform/administrators;exo</value>
    </value-param>
  </init-params>
</component>
```

1.1 Modify the web.xml

- Remove the filter FactoryInitializedFilter
- Remove its corresponding filter mappings (see pattern /connector and /jcr/)
- Add the filters **SetCurrentIdentityFilter** and **RestEncodingFilter** as follow:

```
<filter>
  <filter-name>SetCurrentIdentityFilter</filter-name>
  <filter-class>org.exoplatform.services.security.web.SetCurrentIdentityFilter</filter-class>
</filter>

<filter>
  <filter-name>RestEncodingFilter</filter-name>
    <filter-class>org.exoplatform.services.rest.servlet.RestEncodingFilter</filter-class>
    <init-param>
      <param-name>REQUEST_ENCODING</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
</filter>
```

- Modify the filter mappings as follow:

Replace

```
<filter-mapping>
  <filter-name>ThreadLocalSessionProviderInitializedFilter</filter-name>
  <url-pattern>/private/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>ThreadLocalSessionProviderInitializedFilter</filter-name>
  <url-pattern>/public/*</url-pattern>
</filter-mapping>
```

with

```
<filter-mapping>
  <filter-name>SetCurrentIdentityFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>ThreadLocalSessionProviderInitializedFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>RestEncodingFilter</filter-name>
  <url-pattern>/rest/*</url-pattern>
</filter-mapping>
```

- Replace the SkinListener

Replace

```
<listener>
  <listener-class>org.exoplatform.portal.webui.skin.SkinListener</listener-class>
</listener>
```

with

```
<listener>
  <listener-class>org.exoplatform.portal.webui.skin.SkinConfigListener</listener-class>
</listener>
```

Click here for more details

- Remove the servlet *DisplayJCRContent*
- Remove its corresponding servlet mappings (see pattern /jcr/)

8.4. ECM Migration from 2-0 to 2-0-x

Since ECM 2.0.x, we added property **exo:accessPermission* to drive and view node. So we provided the migration service called PropertiesMigrationService at the location**

ecm/branches/2.0.x/component/migration/2.0.x/properties. This service is used to add property exo:accessPermission* to drive node and view node which didn't contain this property in ECM 2.0.

How to migrate it ? Follow these steps :

Step 1: Compile the source code to create new jar **(exo.ecm.component.migration.2.0.x.properties-2.0.x.jar)** by command:

```
mvn clean install
```

Step 2: Stop server and copy this jar to library.

Step 3: Run server to DriveMigrationService apply the changes.

Step 4: After server has started, stop server again and remove this jar, restart server.

8.5. ECM Migration from 2-1-x to 2-2

8.6. Changing configuration in exo.ecm.web.portal/src/main/java/conf/configuration.xml

Publication function: In ECM 2.1, we can see information in **exo.ecm.component.publication/configuration.xml** :

```
<component>
    <key>org.exoplatform.services.ecm.publication.PublicationService</key>
    <type>org.exoplatform.services.ecm.publication.impl.PublicationServiceImpl</type>
    <component-plugins>
            <component-plugin>
            <name>StaticAndDirect</name>
               <set-method>addPublicationPlugin</set-method>
            <type>org.exoplatform.services.ecm.publication.plugins.staticdirect.StaticAndDirectPublicationPlugin
               <init-params>
                 <values-param>
                    <name>StaticAndDirect</name>
                    <description>This publication lifecycle keeps the content at their original place. Any versi
                    </description>
                 </values-param>
               </init-params>
         </component-plugin>
    </component-plugins>
  </component>
```

And

```
    <external-component-plugins>
       <target-component>org.exoplatform.services.ecm.publication.PublicationService</target-component>
       <component-plugin>
            <name>addPublication</name>
            <set-method>addPublicationPlugin</set-method>
            <type>org.exoplatform.services.ecm.publication.plugins.staticdirect.StaticAndDirectPublicationPlugin
            <init-params>
              <values-param>
                <name>StaticAndDirect</name>
                <description>StaticAndDirect Publication Plugin</description>
              </values-param>
```

```
            </init-params>
        </component-plugin>
    </external-component-plugins>
```

These above informations do not exist any longer in this file in ECM 2.2.

They are moved and changed in a new file **exo.ecm.web.portal/src/main/java/conf/portal/publication-configuration.xml**.

So we must refer this file by adding code in **exo.ecm.web.portal/src/main/java/conf/configuration.xml** like:

```
   <import>war:/conf/ecm/publication-configuration.xml</import>
```

Coverflow and Thumbnails view mode:

In ECM 2.2, we add new service ThumbnailRESTService in **exo.ecm.component.cms** for Coverflow and Thumbnails view mode. To use this service, we declare this class by using new file: **ecm-thumbnail-configuration.xml**:

```
<configuration>
    <component>
        <key>org.exoplatform.services.cms.thumbnail.ThumbnailService</key>
        <type>org.exoplatform.services.cms.thumbnail.impl.ThumbnailServiceImpl</type>
        <init-params>
                <value-param>
                      <name>smallSize</name>
                      <value>32x32</value>
               </value-param>
                <value-param>
                      <name>mediumSize</name>
                      <value>64x64</value>
             </value-param>
               <value-param>
                      <name>bigSize</name>
               <value>300x300</value>
               </value-param>
                <value-param>
                      <name>enable</name>
                      <value>true</value>
                   </value-param>
               <value-param>
                       <name>mimetypes</name>
                <value>image/jpeg;image/png;image/gif;image/bmp</value>
               </value-param>
       </init-params>
    </component>
    <component>
             <type>org.exoplatform.services.cms.thumbnail.impl.ThumbnailRESTService</type>
    </component>
</configuration>
```

Then add the configuration information to **exo.ecm.web.portal/src/main/java/conf/configuration.xml** like:

```
       <import>war:/conf/ecm/ecm-thumbnail-configuration.xml</import>
```

1 Changing version in the configuration information for WorkflowServiceContainer class

In ECM 2.1, we can see in **bonita-configuration.xml** and **jbpm-configuration.xml**:

```
 <object type="org.exoplatform.services.workflow.ProcessesConfig">
```

```
   <fieldname="processLocation"><string>war:/conf/bp</string></field>
   <field name="predefinedProcess">
      <collection type="java.util.HashSet">
         <value><string>/exo.ecm.bp.bonita.payraise-2.1.jar</string></value>
         <value><string>/exo.ecm.bp.bonita.holiday-2.1.jar</string></value>
         <value><string>/exo.ecm.bp.bonita.content.validation-2.1.jar</string></value>
         <value><string>/exo.ecm.bp.bonita.content.backup-2.1.jar</string></value>
      </collection>
   </field>
 </object>
```

And:

```
<object type="org.exoplatform.services.workflow.ProcessesConfig">
   <field name="processLocation"><string>war:/conf/bp</string></field>
   <field name="predefinedProcess">
        <collection type="java.util.HashSet">
         <value><string>/exo.ecm.bp.jbpm.payraise-2.1.jar</string></value>
         <value><string>/exo.ecm.bp.jbpm.holiday-2.1.jar</string></value>
                <value><string>/exo.ecm.bp.jbpm.content.validation-2.1.jar</string></value>
         <value><string>/exo.ecm.bp.jbpm.content.backup-2.1.jar</string></value>
        </collection>
   </field>
</object>
```

So, in ECM 2.2, we must change these files by replacing : **2.1.jar** to **2.2**:

```
<object type="org.exoplatform.services.workflow.ProcessesConfig">
   <fieldname="processLocation"><string>war:/conf/bp</string></field>
   <field name="predefinedProcess">
     <collection type="java.util.HashSet">
        <value><string>/exo.ecm.bp.bonita.payraise-2.2.jar</string></value>
        <value><string>/exo.ecm.bp.bonita.holiday-2.2.jar</string></value>
        <value><string>/exo.ecm.bp.bonita.content.validation-2.2.jar</string></value>
        <value><string>/exo.ecm.bp.bonita.content.backup-2.2.jar</string></value>
    </collection>
   </field>
 </object>
```

And:

```
<object type="org.exoplatform.services.workflow.ProcessesConfig">
   <field name="processLocation"><string>war:/conf/bp</string></field>
   <field name="predefinedProcess">
     <collection type="java.util.HashSet">
        <value><string>/exo.ecm.bp.jbpm.payraise-2.2.jar</string></value>
        <value><string>/exo.ecm.bp.jbpm.holiday-2.2.jar</string></value>
         <value><string>/exo.ecm.bp.jbpm.content.validation-2.2.jar</string></value>
        <value><string>/exo.ecm.bp.jbpm.content.backup-2.2.jar</string></value>
     </collection>
   </field>
</object>
```

1 Supports add references when upload file

Since ECM 2.2, in the File Explorer portlet, you can add references to the file which will be uploaded. By default this function is not mandatory. You can change the value equal true in the parameter **categoryMandatoryWhenFileUpload** to make sure every file will be added categories when uploaded.

You can see and change the configuration if you want in the file **exo.ecm.portlet.ecm/src/main/webapp/WEB-INF/portlet.xml**

```
<preference>
```

```
  <name>categoryMandatoryWhenFileUpload</name>
  <value>false</value>
  <read-only>false</read-only>
</preference>
```

8.7. Add filter for resouces in exo.ecm.web.portal/src/main/webapp/WEB-INF/web.xml

```
<filter-mapping>
  <filter-name>ResourceRequestFilter</filter-name>
  <url-pattern>*.css</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>ResourceRequestFilter</filter-name>
  <url-pattern>*.gif</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>ResourceRequestFilter</filter-name>
  <url-pattern>*.png</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>ResourceRequestFilter</filter-name>
  <url-pattern>*.jpg</url-pattern>
</filter-mapping>
```

1 Add a workspace named gadgets in **exo.ecm.web.portal/src/main/webapp/WEB-INF/conf/jcr/repository-configuration.xml**

```
       <workspace name="gadgets">
       <!-- for system storage -->
         <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
           <properties>
             <property name="source-name" value="jdbcexo"/>
             <property name="dialect" value="hsqldb"/>
             <!-- property name="db-type" value="mysql"/ -->
             <property name="multi-db" value="false"/>
             <property name="update-storage" value="true"/>
             <property name="max-buffer-size" value="200k"/>
             <property name="swap-directory" value="../temp/swap/gadgets"/>
           </properties>
           <value-storages>
             <value-storage id="gadgets" class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValu
               <properties>
                 <property name="path" value="../temp/values/gadgets"/>
               </properties>
               <filters>
                 <filter property-type="Binary"/>
               </filters>
            </value-storage>
           </value-storages>
         </container>
         <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
           <properties>
             <property name="root-nodetype" value="nt:unstructured"/>
             <property name="root-permissions" value="any read;*:/platform/administrators read;*:/platform/admi
           </properties>
         </initializer>
         <cache enabled="true" class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStor
           <properties>
             <property name="max-size" value="20k"/>
             <property name="live-time" value="1h"/>
           </properties>
         </cache>
         <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
           <properties>
             <property name="index-dir" value="../temp/jcrlucenedb/gadgets"/>
           </properties>
          </query-handler>
```

```
          <lock-manager>
            <time-out>15m</time-out><!-- 15min -->
            <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
              <properties>
                <property name="path" value="../temp/lock/gadgets"/>
              </properties>
            </persister>
          </lock-manager>
        </workspace>
```

8.8. ECM Migration from 2-2-x to 2-2-2

8.9. Update a pom.xml

Update the pom.xml at the location web/ecmportal/

In ECM 2.2.2 we changed the pom.xml to make war overlay work better.

```
 <dependentWarExcludes>
   index.jsp,WEB-INF/web.xml,WEB-INF/conf/*.xml,
   WEB-INF/conf/common/*.xml,
   WEB-INF/conf/database/*.xml,
   WEB-INF/conf/database/*.xml,
   WEB-INF/conf/jcr/*.xml,
   WEB-INF/conf/organization/*.xml,
   WEB-INF/conf/portal/*.xml,
   WEB-INF/conf/portal/group/,
   WEB-INF/conf/portal/portal/,
   WEB-INF/conf/portal/user/,
   WEB-INF/classes/locale/navigation/,
   WEB-INF/conf/script/groovy/SkinConfigScript.groovy,
   templates/groovy/webui/component/UIHomePagePortlet.gtmpl,
   templates/skin/webui/component/UIHomePagePortlet/
 </dependentWarExcludes>
```

8.10. Update the application-registry-configuration.xml

Update the application-registry-configuration.xml located at web/ecmportal/src/main/webapp/WEB-INF/conf/portal.

In ECM 2.2.2 we changed the configuration to define the dashboard as the default portlet.

```
<object-param>
    <name>dashboard</name>
    <description>description</description>
    <object type="org.exoplatform.application.registry.ApplicationCategory">
      <field name="name"><string>dashboard</string></field>
      <field name="displayName"><string>Dashboard</string></field>
      <field name="description"><string>Dashboard</string></field>
      <field name="accessPermissions">
        <collection type="java.util.ArrayList" item-type="java.lang.String">
                                      <value><string>*:/platform/users</string></value>
        </collection>
      </field>
      <field  name="applications">
        <collection type="java.util.ArrayList">
          <value>
            <object type="org.exoplatform.application.registry.Application">
              <field name="categoryName"><string>dashboard</string></field>
              <field name="applicationName"><string>DashboardPortlet</string></field>
                          <field name="displayName"><string>Dashboard Portlet</string></field>
```

```
                        <field name="description"><string>Dashboard Portlet</string></field>
             <field name="applicationType"><string>portlet</string></field>
             <field name="applicationGroup"><string>dashboard</string></field>
             <field name="accessPermissions">
                                    <collection type="java.util.ArrayList" item-type="java.lang.String">
                                                          <value><string>*:/platform/users</string
                                    </collection>
                        </field>
           </object>
         </value>
         <value>
           <object type="org.exoplatform.application.registry.Application">
             <field name="categoryName"><string>dashboard</string></field>
             <field name="applicationName"><string>GadgetPortlet</string></field>
                        <field name="displayName"><string>Gadget Wrapper Portlet</string></field>
                        <field name="description"><string>Gadget Wrapper Portlet</string></field>
             <field name="applicationType"><string>portlet</string></field>
             <field name="applicationGroup"><string>dashboard</string></field>
             <field name="accessPermissions">
                                    <collection type="java.util.ArrayList" item-type="java.lang.String">
                                                          <value><string>*:/platform/users</string
                                    </collection>
                        </field>
           </object>
         </value>
       </collection>
     </field>
   </object>
 </object-param>
```

8.11. ECM Migration from DMS 2-3 to DMS 2.4

8.12. New features with Symlink and Taxonomy Management

- In order to be able to use a virtual access to a document, we need node type **exo:simlink*. In addition, for** Manage Taxonomy**, we use new nodetype** exo:taxonomyLink* in file **nodetypes-config-extended.xml** in **dms/core/component/cms/src/main/java/conf/portal**:

```
<nodeType name="exo:symlink" isMixin="false" hasOrderableChildNodes="false" primaryItemName="exo:primaryType">
 <supertypes>
  <supertype>nt:hierarchyNode</supertype>
 </supertypes>
 <propertyDefinitions>
  <propertyDefinition name="exo:workspace" requiredType="String" autoCreated="false" mandatory="false"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
  <propertyDefinition name="exo:uuid" requiredType="String" autoCreated="false" mandatory="true"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
  <propertyDefinition name="exo:primaryType" requiredType="Name" autoCreated="false" mandatory="true"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
 </propertyDefinitions>
</nodeType>

<nodeType name="exo:taxonomyLink" isMixin="false" hasOrderableChildNodes="false" primaryItemName="exo:primaryTyp
 <supertypes>
  <supertype>exo:symlink</supertype>
 </supertypes>
</nodeType>
```

- Please refer to Symlink>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+Symlink and Manage Taxonomy>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+new+Taxonomy+Tree to get the details of these implementations

1 Using new workspace dms-system

Instead of storing in system workspace, from DMS-2.4, we use **dms-system** to store **taxonomy**, **templates**, **query**, **meta data**, **scripts**...

- In file **repository-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/jcr**:

add new workspace **dms-system**

```
<workspace name="dms-system">
  <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
 <properties>
   <property name="source-name" value="jdbcexo"/>
   <property name="dialect" value="hsqldb"/>
   <property name="multi-db" value="false"/>
   <property name="max-buffer-size" value="200k"/>
   <property name="swap-directory" value="../temp/swap/dms-system"/>
 </properties>
 <value-storages>
   <value-storage id="dms-system" class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage
  <properties>
    <property name="path" value="../temp/values/dms-system"/>
  </properties>
  <filters>
    <filter property-type="Binary"/>
  </filters>
   </value-storage>
 </value-storages>
  </container>
  <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
 <properties>
   <property name="root-nodetype" value="nt:unstructured"/>
   <property name="root-permissions" value="any read;*:/platform/administrators read;*:/platform/administrators
 </properties>
  </initializer>
  <cache enabled="true" class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCache
 <properties>
   <property name="max-size" value="20k"/>
   <property name="live-time" value="1h"/>
 </properties>
  </cache>
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
 <properties>
   <property name="indexDir" value="../temp/jcrlucenedb/dms-system"/>
   <property name="support-highlighting" value="true"/>
   <property name="excerptprovider-class" value="org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTML
 </properties>
  </query-handler>
  <lock-manager>
 <time-out>15m</time-out>
 <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
   <properties>
  <property name="path" value="../temp/lock/dms-system"/>
   </properties>
 </persister>
  </lock-manager>
</workspace>
```

- To get name of this workspace, we add new configuration file **dms-common-configuration.xml** at the location

dms/core/web/portal/.../WEB-INF/conf/dms

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
  <external-component-plugins>
  <target-component>org.exoplatform.services.cms.impl.DMSConfiguration</target-component>
    <component-plugin>
      <name>dmsconfiguration.plugin</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.cms.impl.DMSRepositoryConfiguration</type>
      <description>DMS Repository configuration</description>
      <init-params>
       <value-param>
         <name>repository</name>
         <value>repository</value>
       </value-param>
       <value-param>
         <name>systemWorkspace</name>
         <value>dms-system</value>
       </value-param>
      </init-params>
    </component-plugin>
 </external-component-plugins>
</configuration>
```

- In file **dms-drives-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms** We change the location of drive **Categorized Documents** and add new drive **DMS Administration** to control data in this new workspace by modifying/appending xml code.

```
...
<object-param>
 <name>Categorized Documents</name>
 <description>Categorized Documents</description>
 <object type="org.exoplatform.services.cms.drives.DriveData">
   <field  name="name"><string>Categorized Documents Center</string></field>
   <field  name="repository"><string>repository</string></field>
   <field  name="workspace"><string>dms-system</string></field>
   <field  name="permissions"><string>*:/platform/administrators</string></field>
   <field  name="homePath"><string>/exo:ecm/exo:taxonomyTrees/storage</string></field>
   <field  name="icon"><string></string></field>
   <field  name="views">
  <string>taxonomy-list,taxonomy-icons</string>
   </field>
   <field  name="viewPreferences"><boolean>true</boolean></field>
   <field  name="viewNonDocument"><boolean>true</boolean></field>
   <field  name="viewSideBar"><boolean>true</boolean></field>
   <field  name="showHiddenNode"><boolean>false</boolean></field>
   <field  name="allowCreateFolder"><string>Both</string></field>
 </object>
</object-param>
...
<object-param>
 <name>DMS Administration</name>
 <description>DMS system data area</description>
 <object type="org.exoplatform.services.cms.drives.DriveData">
   <field  name="name"><string>DMS Administration</string></field>
   <field  name="repository"><string>repository</string></field>
   <field  name="workspace"><string>dms-system</string></field>
   <field  name="permissions"><string>*:/platform/administrators</string></field>
   <field  name="homePath"><string>/</string></field>
   <field  name="icon"><string></string></field>
   <field  name="views">
  <string>icon-view, simple-view, admin-view, cover-flow</string>
   </field>
   <field  name="viewPreferences"><boolean>false</boolean></field>
   <field  name="viewNonDocument"><boolean>true</boolean></field>
   <field  name="viewSideBar"><boolean>true</boolean></field>
   <field  name="showHiddenNode"><boolean>false</boolean></field>
   <field  name="allowCreateFolder"><string>Both</string></field>
 </object>
</object-param>
```

- We move all data of **taxonomy**, **templates**, **query**, **meta data**, **scripts**... from **system** workspace to new **dms-system** workspace but we still keep data of folksonomy in **system** workspace. These changes are configurated in file **dms-system-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
 <external-component-plugins>
  <target-component>org.exoplatform.services.jcr.ext.hierarchy.NodeHierarchyCreator</target-component>
  <component-plugin>
   <name>addPaths</name>
   <set-method>addPlugin</set-method>
   <type>org.exoplatform.services.jcr.ext.hierarchy.impl.AddPathPlugin</type>
   <init-params>
    <object-param>
     <name>cms.configuration</name>
     <description>configuration for the cms path</description>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig">
      <field name="repository">
       <string>repository</string>
      </field>
      <field name="workspaces">
       <collection type="java.util.ArrayList">
        <value>
         <string>dms-system</string>
        </value>
       </collection>
      </field>
      <field name="jcrPaths">
       <collection type="java.util.ArrayList">
        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
           <string>exoECMSystemPath</string>
          </field>
          <field name="path">
           <string>/exo:ecm</string>
          </field>
          <field name="permissions">
           <collection type="java.util.ArrayList">
            <value>
             <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
              <field name="identity">
               <string>*:/platform/administrators</string>
              </field>
              <field name="read">
               <string>true</string>
              </field>
              <field name="addNode">
               <string>true</string>
              </field>
              <field name="setProperty">
               <string>true</string>
              </field>
              <field name="remove">
               <string>true</string>
              </field>
             </object>
            </value>
            <value>
 .....
```

See complete file: "dms-system-configuration.xml"

1 Create default taxonomy tree to [Manage Taxonomy>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+new+Taxonomy+Tree]

- From DMS 2-4, we remove old taxonomy and create a taxonomy tree named **System**. This tree is defined in

dms-taxonomies-configuration.xml at the location **dms/core/web/portal/.../WEB-INF/conf/dms/** and is automatically created when tomcat runs

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>

  <component>
   <key>org.exoplatform.services.cms.taxonomy.TaxonomyService</key>
   <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyServiceImpl</type>
 </component>

 <external-component-plugins>
    <target-component>org.exoplatform.services.cms.taxonomy.TaxonomyService</target-component>
    <component-plugin>
     <name>predefinedTaxonomyPlugin</name>
      <set-method>addTaxonomyPlugin</set-method>
      <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyPlugin</type>
      <init-params>
       <value-param>
          <name>autoCreateInNewRepository</name>
          <value>true</value>
        </value-param>
        <value-param>
         <name>repository</name>
         <value>repository</value>
        </value-param>
        <value-param>
         <name>treeName</name>
         <value>System</value>
        </value-param>
         <object-param>
           <name>permission.configuration</name>
           <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig">
             <field name="taxonomies">
               <collection type="java.util.ArrayList">
                 <value>
                   <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Taxonomy">
                     <field name="permissions">
                <collection type="java.util.ArrayList">
                   <value>
                     <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission">
                        <field name="identity"><string>any</string></field>
                        <field name="read"><string>true</string></field>
                        <field name="addNode"><string>false</string></field>
                        <field name="setProperty"><string>false</string></field>
                        <field name="remove"><string>false</string></field>
                      </object>
                   </value>
                   <value>
                     <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission">
                      <field name="identity"><string>*:/platform/administrators</string></field>
                        <field name="read"><string>true</string></field>
                        <field name="addNode"><string>true</string></field>
                        <field name="setProperty"><string>true</string></field>
                        <field name="remove"><string>true</string></field>
                     </object>
                   </value>
                 </collection>
               </field>
                 </object>
               </value>
             </collection>
           </field>
         </object>
        </object-param>
        <object-param>
          <name>predefined.actions</name>
          <description>description</description>
          <object type="org.exoplatform.services.cms.actions.impl.ActionConfig">
            <field  name="actions">
              <collection type="java.util.ArrayList">
                <value>
                  <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$TaxonomyAction">
                    <field  name="type"><string>exo:taxonomyAction</string></field>
                    <field  name="name"><string>taxonomyAction</string></field>
```

```
                    <field  name="description"><string></string></field>
                    <field  name="homePath"><string>dms-system:/exo:ecm/exo:taxonomyTrees/storage/System</stri
                    <field  name="targetWspace"><string>collaboration</string></field>
                    <field  name="targetPath"><string>/Documents</string></field>
                    <field  name="lifecyclePhase"><string>read</string></field>
                    <field  name="roles"><string>*:/platform/administrators</string></field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </object-param>
      <object-param>
...
```

See complete file: ^dms-taxonomies-configuration.xml

8.13. Filter all action in File Explorer, ECM Admin

- In DMS 2-4 portlet **ecm** is splited to 4 sub-projects:

1. **core/portlet/ecm/core/main**: Only the java part of the old **core/portlet/ecm** project. The pom of this project creates a jar that will be added in the directory **ecm.war/WEB-INF/lib**
2. **core/portlet/ecm/core/web**: Only the web part of the old core/portlet/ecm project. The pom of this project creates a war that will be the **ecm.war**
3. **core/portlet/ecm/ext/main**: Only the java part of the current extensions. The pom of this project creates a jar that will be added in the directory **ecm.war/WEB-INF/lib**
4. **core/portlet/ecm/ext/config**: Only the configuration of those extensions. The pom of this project creates a jar that will be added in the directory **tomcat/lib**

- Now all action in File Explorer and ECM Admin are filtered for each view. All actions are registered and manage by **org.exoplatform.webui.ext.UIExtensionManager**. We configure actions in **dms-ext-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms/**. A complete example: ^dms-ext-configuration.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>
  <component>
  <key>org.exoplatform.webui.ext.UIExtensionManager</key>
  <type>org.exoplatform.webui.ext.impl.UIExtensionManagerImpl</type>
  <component-plugins>
   <component-plugin>
    <name>Add Actions</name>
    <set-method>registerUIExtensionPlugin</set-method>
    <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
    <init-params>
     <object-param>
      <name>AddFolder</name>
      <object type="org.exoplatform.webui.ext.UIExtension">
       <field name="type"><string>${typeOfAction}</string></field>
       <field name="name"><string>${ActionName}</string></field>
       <field name="component"><string>${ActionHandle}</string></field>
      </object>
...
```

Please refer to http://wiki.exoplatform.com/xwiki/bin/view/ECM/UI+Extension+Framework

1 Gadget for last edited documents **LastEditedDocuments** gadget displays last 5 edited documents with document name and edited time. When user click on the document, its content will be displayed in File Explorer

- In **GetEditedDocumentRESTService.class** (exo.ecm.dms.core.component.cms-2.4.jar) we implement Rest Service to get these document.
- Add new file **LastEditedDocuments.xml** in **src/main/webapp/gadgets** for displaying gadget

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
 <ModulePrefs author="eXoPlatform"
  title="Last edited documents"
  directory_title="Last edited documents"
  title_url="http://www.exoplatform.org"
  description="The last edited documents"
  height="230">
 </ModulePrefs>
 <Content type="html">
 <![CDATA[

  <script type="text/javascript" src="/eXoDMSGadgets/javascript/script.js"></script>

  <style type="text/css">
   body{
    font: 11px Tahoma,Verdana,Arial,Helvetica,sans-serif;
    height:100%;
    margin:0 auto;
    padding:0;
   }

   a {
    text-decoration: none;
   }

   ...
   </style>
    <script type="text/javascript">
   function LastEditedDocument() {

   }

   var siteUrl = 'http://localhost:8080/portal/rest/presentation/document/edit/repository?showItems=5';
   var accessUrl = 'http://localhost:8080/portal/private/classic/contentmanagement/fileexplorer?portal:component

   LastEditedDocument.prototype.makeRequest = function(url, callback) {
    var params = {};
    params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
    params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.JSON;
    var ts = new Date().getTime();
     var sep = "?";
     var refreshInterval = 60;
     if (refreshInterval && refreshInterval > 0) {
       ts = Math.floor(ts / (refreshInterval * 1000));
     }
     if (url.indexOf("?") > -1) {
       sep = "&";
     }
     url = [ url, sep, "nocache=", ts ].join("");
    gadgets.io.makeRequest(url, callback, params);
   }

   LastEditedDocument.prototype.init = function() {
    lastEditedDocument.makeRequest(siteUrl,lastEditedDocument.displayValue);
   }

   LastEditedDocument.prototype.displayValue = function(resp) {
    var data = resp.data.lstDocNode;
    var editedDocList = _gel("ListEditDocumentNode");
    var displayTime = new DisplayTime();
```

```
    for (var i = 0; i < data.length; i++) {
     var doc = data[i];
     var AppClass = document.createElement('div');
     var showHTML = "<a target='_top' href='";
     showHTML+= accessUrl;
     showHTML+= "&repository=repository";
     showHTML+= "&drive=";
     showHTML+= doc.driveName;
     showHTML+= "&path=";
     showHTML+= doc.path;
     showHTML+= "'>";
     showHTML+=doc.name;
     showHTML+="</a>";
     showHTML+="<div class='DateTime'>";
     showHTML+="";
     showHTML+=displayTime.timeToPrettyString(doc.dateEdited);
     showHTML+="</div>";
     AppClass.innerHTML = showHTML;
     AppClass.className = 'ItemIcon';
     editedDocList.appendChild(AppClass);
    }
    var title = _gel("BgTitle");
    if (data && (data.length > 0) && title) {
     editedDocList.parentNode.removeChild(title);
    }
    gadgets.window.adjustHeight();
   }

   var lastEditedDocument = new LastEditedDocument();
   gadgets.util.registerOnLoadHandler(lastEditedDocument.init);
  </script>

  <body>

   <div class="ContentGadGet">
    <div class="BgLeftTitleContent">
     <div class="BgRightTitleContent">
      <div class="BgCentertTitleContent">
       <div class="IconArrow"><span></span></div>
       <div class="TextTitle">Last Edited Documents</div>
       <div class="ClearRight"><span></span></div>
      </div>
     </div>
    </div>
    <div class="BgRightContent">
     <div class="BgLeftContent">
      <div class="BgCenterContent">
       <div id="BgTitle" class="BgTitle">
        <div class="NotAvailableItemIcon">
         Not available document!
        </div>
       </div>
       <div id="ListEditDocumentNode" class="ListEditDocumentNode">
       </div>
      </div>
     </div>
    </div>
    <div class="BgLeftFooterContent">
     <div class="BgRightFooterContent">
      <div class="BgCenterFooterContent"><span></span></div>
     </div>
    </div>
   </div>
  </body>
 ]]>
 </Content>
</Module>
```

8.14. Set up size of uploading file

- When we upload file in File Explorer, we need limit size of file. Maximum size of file is set up in

portlet-preferences.xml at the location **dms/core/web/portal/.../WEB-INF/conf/portal/group/platform/users**. In this file, we set 30M by default for file size

```
<portlet-preferences>
...
 <preference>
  <name>uploadFileSizeLimitMB</name>
  <value>30</value>
  <read-only>false</read-only>
 </preference>
...
</portlet-preferences>
```

8.15. Migrate data from DMS 2.3 to DMS 2.5

8.16. Migration for workspace and resource bundle

Since DMS version 2.5 or higher, we used dms-system workspace as a location to store all data of DMS(templates, drives, scripts,...). So when we want to upgrade DMS from 2.3 to 2.5 and keep all data as before then we have to create this workspace.

How to do it?

- Compile the source code at /dms.version}/core/component/migration/2.5/workspaces to create new jar **(exo.ecm.dms.core.component.migration.2.5.workspaces-dms.version}.jar)** by command:

```
mvn clean install
```

or simply by copy>http:--wiki.exoplatform.com-xwiki-bin-download-ECM-ECM+Migration+from+DMS+2%2D3+to+DMS+2%2D5-e the jar file which complied and attached to this article

-
  - Copy this jar to server's library(For ex: the "/lib" folder in tomcat server).
  - Stop and run server to let this jar apply the changes. You can see some exceptions in the console because the new configuration cannot found the dms-system workspace to init data. Don't worry, let it run to last step, after that stop and start server again please, everything will be fine
  - Remove this jar.

1 Migration for data in dms-system workspace In the older version of DMS(before 2.5), we used system workspace to store all data of DMS, that is mean since DMS 2.5 those data are useless and need to be migrated to dms-system workspace to make it effect to our system.

To do this thing, we have to migrate workspace and resource bundle which described above first. After that follow step by step

- Compile the source code at /dms.version}/core/component/migration/2.5/datas to create new jar

(exo.ecm.dms.core.component.migration.2.5.datas-dms.version}.jar) by command:

```
mvn clean install
```

or simply by copy the jar file which complied and attached to this article

-
  - Copy this jar to server's library(For ex: the "/lib" folder in tomcat server).
  - Stop and run server to let this jar apply the changes.
  - Remove this jar.

8.17. ECM Migration from ECM 2-2-x to DMS 2-3

Since ECM 2.3, we split ECM to 3 products as DMS, Workflow and WCM. So, how can we upgrade from 2.2 to 2.3 ? You can follow these steps to get successful upgrade. #info("For now, we have the ECM suite which contains 3 products as

```
ECM(Enterprise Content Management)
 |
 |__DMS(Document Management System)
 |    |__core
 |    |__ext
 |        |__contentvalidation
 |
 |__Workflow
 |
 |__WCM(Web Content Management)
```

")

1 Migration for drives path

Since ECM 2.3, the DriveMigrationService was available at the location **ecm/dms/trunk/core/component/migration/2.3/drives**. This service is used to rename the old drive which contains invalid characters and prevent the WARNING messages at the console.

How to do it?

- Compile the source code to create new jar **(exo.ecm.dms.core.component.migration.2.3.drives-2.3.jar)** by command:

```
mvn clean install
```

-
  - Stop server and copy this jar to library.
  - Run server DriveMigrationService to apply changes.
  - After server has started, stop server again and remove this jar, restart server.

8.18. How to use for Workflow Publication Plugin ?

Since ECM 2.3, the Workflow Publication Plugin was available at the location **ecm/dms/trunk/ext/contentvalidation/component/workflowPublication/**. This plugin is used to publish a content of documents. It breaks a work process down into tasks. See more details about Workflow:Workflow Publication Plugin.

How to do it?

- Compile the source code to create new jar **(exo.ecm.dms.ext.contentvalidation.component.workflowPublication-2.3.jar)** by command:

```
mvn clean install
```

-
  - Stop server and copy this jar to server library (for example: omcatib)
  - Run server to apply the changes.

Here are the structure of this jar

```
exo.ecm.dms.ext.contentvalidation.component.workflowPublication-2.3.jar
 |
 |__conf
 |  |
 |  |__nodetypes-workflow-publication-config.xml
 |  |
 |  |__workflow-templates-configuration.xml
 |  |
 |  |__conf.portal
 |     |
 |     |__configuration.xml
 |
 |__...
 |
 |__...
```

- **configuration.xml**: This file is used to plug a new plugin (Workflow Publication Plugin). It will specify to two files: *nodetypes-workflow-publication-config.xml*, *workflow-templates-configuration.xml* to configure for this plugin

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>

  <component>
    <key>org.exoplatform.services.ecm.publication.PublicationService</key>
    <type>org.exoplatform.services.ecm.publication.impl.PublicationServiceImpl</type>
  </component>

  <component>
    <key>org.exoplatform.services.ecm.publication.PublicationPresentationService</key>
    <type>org.exoplatform.services.ecm.publication.impl.PublicationPresentationServiceImpl</type>
  </component>

  <external-component-plugins>
    <target-component>org.exoplatform.services.jcr.RepositoryService</target-component>
    <component-plugin>
      <name>add.nodeType</name>
```

```
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.jcr.impl.AddNodeTypePlugin</type>
      <priority>122</priority>
      <init-params>
        <values-param>
          <name>autoCreatedInNewRepository</name>
          <description>Node types configuration file</description>
          <value>jar:/conf/nodetypes-workflow-publication-config.xml</value>
        </values-param>
       </init-params>
     </component-plugin>
  </external-component-plugins>

  <external-component-plugins>
    <target-component>org.exoplatform.services.ecm.publication.PublicationService</target-component>
    <component-plugin>
      <name>Workflow</name>
      <set-method>addPublicationPlugin</set-method>
      <type>org.exoplatform.services.ecm.publication.plugins.workflow.WorkflowPublicationPlugin</type>
      <description>Workflow Publication</description>
      <init-params>
        <value-param>
          <name>validator</name>
          <value>*:/platform/administrators</value>
        </value-param>
        <value-param>
          <name>to_workspace</name>
          <value>collaboration</value>
        </value-param>
        <value-param>
          <name>destPath</name>
          <value>/Documents/Live</value>
        </value-param>
        <value-param>
          <name>destPath_currentFolder</name>
          <value>true</value>
        </value-param>
        <value-param>
          <name>isEditable</name>
          <value>false</value>
        </value-param>
        <value-param>
          <name>backupWorkspace</name>
          <value>backup</value>
        </value-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
  <import>jar:/conf/workflow-templates-configuration.xml</import>
</configuration>
```

- **nodetypes-workflow-publication-config.xml**: This file is used to configure a new nodetype

```
<nodeTypes xmlns:nt="http://www.jcp.org/jcr/nt/1.0" xmlns:mix="http://www.jcp.org/jcr/mix/1.0"
  xmlns:jcr="http://www.jcp.org/jcr/1.0">

  <nodeType name="publication:workflowPublication" isMixin="true" hasOrderableChildNodes="false" primaryItemName
    <supertypes>
      <supertype>publication:publication</supertype>
    </supertypes>
    <propertyDefinitions>
      <propertyDefinition name="publication:validator" requiredType="String" autoCreated="true" mandatory="true"
        onParentVersion="COPY" protected="false" multiple="false">
        <valueConstraints/>
        <defaultValues>
          <defaultValue>*</defaultValue>
        </defaultValues>
      </propertyDefinition>
      <propertyDefinition name="publication:businessProcess" requiredType="String" autoCreated="true" mandatory=
        onParentVersion="COPY" protected="true" multiple="false">
        <valueConstraints/>
        <defaultValues>
          <defaultValue>content-publishing</defaultValue>
```

```
        </defaultValues>
      </propertyDefinition>
      <propertyDefinition name="publication:backupPath" requiredType="String" autoCreated="false" mandatory="tru
        <valueConstraints/>
      </propertyDefinition>
    </propertyDefinitions>
  </nodeType>

  <nodeType name="publication:workflowAction" isMixin="false" hasOrderableChildNodes="false" primaryItemName="">
    <supertypes>
      <supertype>exo:businessProcessAction</supertype>
    </supertypes>
    <propertyDefinitions>
      <propertyDefinition name="exo:validator" requiredType="String" autoCreated="false" mandatory="true"
        onParentVersion="COPY" protected="false" multiple="false"/>
      <propertyDefinition name="exo:businessProcess" requiredType="String" autoCreated="true" mandatory="true"
        onParentVersion="COPY" protected="true" multiple="false">
        <valueConstraints/>
        <defaultValues>
          <defaultValue>content-publishing</defaultValue>
        </defaultValues>
      </propertyDefinition>
    </propertyDefinitions>
  </nodeType>

</nodeTypes>
```

- **workflow-templates-configuration.xml**: This file is used to decide the template of publication : workflowAction. It will specify the dialog as well as view for that template.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <component>
    <key>org.exoplatform.services.cms.templates.TemplateService</key>
    <type>org.exoplatform.services.cms.templates.impl.TemplateServiceImpl</type>
  </component>

  <external-component-plugins>
    <target-component>org.exoplatform.services.cms.templates.TemplateService</target-component>
    <component-plugin>
      <name>addTemplates</name>
      <set-method>addTemplates</set-method>
      <type>org.exoplatform.services.cms.templates.impl.TemplatePlugin</type>
      <init-params>
        <value-param>
          <name>autoCreateInNewRepository</name>
          <value>true</value>
        </value-param>
        <value-param>
          <name>storedLocation</name>
          <value>jar:/resources/templates/workflowAction</value>
        </value-param>
        <value-param>
          <name>repository</name>
          <value>repository</value>
        </value-param>
        <object-param>
          <name>template.configuration</name>
          <description>configuration for the localtion of templates to inject in jcr</description>
          <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig">
            <field name="nodeTypes">
              <collection type="java.util.ArrayList">
                <value>
                  <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType">
                    <field name="nodetypeName"><string>publication:workflowAction</string></field>
                    <field name="documentTemplate"><boolean>false</boolean></field>
                    <field name="label"><string>Workflow Publication Action</string></field>
                    <field name="referencedView">
                      <collection type="java.util.ArrayList">
                        <value>
                          <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                            <field name="templateFile"><string>/workflowActionView.gtmpl</string></field>
```

```
                          <field name="roles"><string>*</string></field>
                        </object>
                      </value>
                    </collection>
                  </field>
                  <field name="referencedDialog">
                    <collection type="java.util.ArrayList">
                      <value>
                        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                          <field name="templateFile"><string>/workflowActionDialog.gtmpl</string></field>
                          <field name="roles"><string>*</string></field>
                        </object>
                      </value>
                    </collection>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>

</configuration>
```

1 How to access a drive directly? Since DMS 2.3, we support 4 ways to access File Explorer as "Selection", "Social", "Personal" and "Jailed". This means, when you switch to use DSM 2.3 you should modify the portlet.xml file to make it work well.

```
  <preference>
    <name>usecase</name>
    <value>selection</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>driveName</name>
    <value>Collaboration Center</value>
    <read-only>false</read-only>
  </preference>
```

By default the usecase is "selection" and you can select any drive to access File Explorer, in this case the driveName preference is not used. In the other cases, you should specify the driveName to access directly to the File Explorer. See more details in How to use access directly FE drive

8.19. How to make BC publication aware?

When you create an instance of Content Browser, all documents are listed under the folder. But since DMS 2.3 we supported one more preference and you can decide which kind of documents should be displayed, publised documents only or every kind of documents.

You can modify something in the portlet.xml file at location core/portlet/ecm/src/main/webapp/WEB-INF/ to use this new function.

```
<preference>
   <name>isAllowPublish</name>
   <value>false</value>
   <read-only>false</read-only>
</preference>
```

See more detail in Migration for CB_Migration+for+CB

1 How to specify the root location of search in BC portlet? When you make a search in BC portlet before, you can not specify the search location. So, we always execute this function at the current location. Since DMS 2.3, we improved this one and make it more powerful by support search with another one. Users can choose the place where they want to execute the search.

How can we do? Make it easy by add preferences to portlet.xml file.

```
<!--
  Enable search root location
-->
 <preference>
   <name>enableSearch</name>
   <value>false</value>
   <read-only>false</read-only>
 </preference>
<!--
 Specify the search root location. This preference affect only when enableSearch is true
 Notice: Have to specify the workspace name
-->
 <preference>
   <name>searchLocation</name>
   <value>collaboration:/Documents/Live</value>
   <read-only>false</read-only>
 </preference>
```

Anyway, you can refer to this page to see more details.

8.20. How to enable GadgetRegistryService?

In the file /exo.ecm.dms.core.web.portal/src/main/webapp/WEB-INF/conf/portal/application-registry-configuration.xml You have to add the configuration for GadgetRegistryService.

```
 <component>
   <key>org.exoplatform.application.gadget.GadgetRegistryService</key>
   <type>org.exoplatform.application.gadget.jcr.GadgetRegistryServiceImpl</type>
   <init-params>
    <properties-param>
      <name>developerInfo</name>
      <description>The group that is allowed to develop gadgets</description>
      <property name="developer.group" value="/platform/administrators"></property>
    </properties-param>
   </init-params>
 </component>
```

Of course, you can change the value of developer.group to make it correspond with your data.

8.21. Changes between DMS 2.3 and 2.5

8.22. New features with Symlink and Taxonomy Management

- In order to be able to use a virtual access to a document, we need node type **exo:simlink*. In addition, for** Manage Taxonomy**, we use new nodetype** exo:taxonomyLink* in file **nodetypes-config-extended.xml** in **dms/core/component/cms/src/main/java/conf/portal**:

```
<nodeType name="exo:symlink" isMixin="false" hasOrderableChildNodes="false" primaryItemName="exo:primaryType">
 <supertypes>
  <supertype>nt:hierarchyNode</supertype>
 </supertypes>
 <propertyDefinitions>
  <propertyDefinition name="exo:workspace" requiredType="String" autoCreated="false" mandatory="false"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
  <propertyDefinition name="exo:uuid" requiredType="String" autoCreated="false" mandatory="true"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
  <propertyDefinition name="exo:primaryType" requiredType="Name" autoCreated="false" mandatory="true"
   onParentVersion="COPY" protected="false" multiple="false">
  <valueConstraints/>
  </propertyDefinition>
 </propertyDefinitions>
</nodeType>

<nodeType name="exo:taxonomyLink" isMixin="false" hasOrderableChildNodes="false" primaryItemName="exo:primaryTyp
 <supertypes>
  <supertype>exo:symlink</supertype>
 </supertypes>
</nodeType>
```

- Please refer to Symlink>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+Symlink and Manage Taxonomy>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+new+Taxonomy+Tree to get the details of these implementations

1 Using new workspace dms-system

Instead of storing in system workspace, from DMS-2.4, we use **dms-system** to store **taxonomy**, **templates**, **query**, **meta data**, **scripts**...

- In file **repository-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/jcr**:

add new workspace **dms-system**

```
<workspace name="dms-system">
  <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
 <properties>
   <property name="source-name" value="jdbcexo"/>
   <property name="dialect" value="hsqldb"/>
   <property name="multi-db" value="false"/>
   <property name="max-buffer-size" value="200k"/>
   <property name="swap-directory" value="../temp/swap/dms-system"/>
 </properties>
 <value-storages>
   <value-storage id="dms-system" class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage
  <properties>
    <property name="path" value="../temp/values/dms-system"/>
  </properties>
  <filters>
    <filter property-type="Binary"/>
  </filters>
   </value-storage>
 </value-storages>
  </container>
  <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
 <properties>
   <property name="root-nodetype" value="nt:unstructured"/>
   <property name="root-permissions" value="any read;*:/platform/administrators read;*:/platform/administrators
 </properties>
  </initializer>
  <cache enabled="true" class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCache
 <properties>
   <property name="max-size" value="20k"/>
```

```
   <property name="live-time" value="1h"/>
 </properties>
  </cache>
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
 <properties>
   <property name="indexDir" value="../temp/jcrlucenedb/dms-system"/>
   <property name="support-highlighting" value="true"/>
   <property name="excerptprovider-class" value="org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTML
 </properties>
  </query-handler>
  <lock-manager>
 <time-out>15m</time-out>
 <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
   <properties>
  <property name="path" value="../temp/lock/dms-system"/>
   </properties>
 </persister>
  </lock-manager>
</workspace>
```

- To get name of this workspace, we add new configuration file **dms-common-configuration.xml** at the location

dms/core/web/portal/.../WEB-INF/conf/dms

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
  <external-component-plugins>
  <target-component>org.exoplatform.services.cms.impl.DMSConfiguration</target-component>
    <component-plugin>
      <name>dmsconfiguration.plugin</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.cms.impl.DMSRepositoryConfiguration</type>
      <description>DMS Repository configuration</description>
      <init-params>
       <value-param>
         <name>repository</name>
         <value>repository</value>
       </value-param>
       <value-param>
         <name>systemWorkspace</name>
         <value>dms-system</value>
       </value-param>
      </init-params>
    </component-plugin>
 </external-component-plugins>
</configuration>
```

- In file **dms-drives-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms** We change the location of drive **Categorized Documents** and add new drive **DMS Administration** to control data in this new workspace by modifying/appending xml code.

```
...
<object-param>
 <name>Categorized Documents</name>
 <description>Categorized Documents</description>
 <object type="org.exoplatform.services.cms.drives.DriveData">
   <field  name="name"><string>Categorized Documents Center</string></field>
   <field  name="repository"><string>repository</string></field>
   <field  name="workspace"><string>dms-system</string></field>
   <field  name="permissions"><string>*:/platform/administrators</string></field>
   <field  name="homePath"><string>/exo:ecm/exo:taxonomyTrees/storage</string></field>
   <field  name="icon"><string></string></field>
   <field  name="views">
  <string>taxonomy-list,taxonomy-icons</string>
   </field>
   <field  name="viewPreferences"><boolean>true</boolean></field>
   <field  name="viewNonDocument"><boolean>true</boolean></field>
```

```
   <field  name="viewSideBar"><boolean>true</boolean></field>
   <field  name="showHiddenNode"><boolean>false</boolean></field>
   <field  name="allowCreateFolder"><string>Both</string></field>
 </object>
</object-param>
...
<object-param>
 <name>DMS Administration</name>
 <description>DMS system data area</description>
 <object type="org.exoplatform.services.cms.drives.DriveData">
   <field  name="name"><string>DMS Administration</string></field>
   <field  name="repository"><string>repository</string></field>
   <field  name="workspace"><string>dms-system</string></field>
   <field  name="permissions"><string>*:/platform/administrators</string></field>
   <field  name="homePath"><string>/</string></field>
   <field  name="icon"><string></string></field>
   <field  name="views">
  <string>icon-view, simple-view, admin-view, cover-flow</string>
   </field>
   <field  name="viewPreferences"><boolean>false</boolean></field>
   <field  name="viewNonDocument"><boolean>true</boolean></field>
   <field  name="viewSideBar"><boolean>true</boolean></field>
   <field  name="showHiddenNode"><boolean>false</boolean></field>
   <field  name="allowCreateFolder"><string>Both</string></field>
 </object>
</object-param>
```

- We move all data of **taxonomy**, **templates**, **query**, **meta data**, **scripts**... from **system** workspace to new **dms-system** workspace but we still keep data of folksonomy in **system** workspace. These changes are configurated in file **dms-system-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
 <external-component-plugins>
  <target-component>org.exoplatform.services.jcr.ext.hierarchy.NodeHierarchyCreator</target-component>
  <component-plugin>
   <name>addPaths</name>
   <set-method>addPlugin</set-method>
   <type>org.exoplatform.services.jcr.ext.hierarchy.impl.AddPathPlugin</type>
   <init-params>
    <object-param>
     <name>cms.configuration</name>
     <description>configuration for the cms path</description>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig">
      <field name="repository">
       <string>repository</string>
      </field>
      <field name="workspaces">
       <collection type="java.util.ArrayList">
        <value>
         <string>dms-system</string>
        </value>
       </collection>
      </field>
      <field name="jcrPaths">
       <collection type="java.util.ArrayList">
        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
           <string>exoECMSystemPath</string>
          </field>
          <field name="path">
           <string>/exo:ecm</string>
          </field>
          <field name="permissions">
           <collection type="java.util.ArrayList">
            <value>
             <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
              <field name="identity">
               <string>*:/platform/administrators</string>
```

```
              </field>
              <field name="read">
               <string>true</string>
              </field>
              <field name="addNode">
               <string>true</string>
              </field>
              <field name="setProperty">
               <string>true</string>
              </field>
              <field name="remove">
               <string>true</string>
              </field>
             </object>
            </value>
            <value>
.....
```

See complete file "dms-system-configuration.xml"

1 Create default taxonomy tree to [Manage Taxonomy>http:--wiki.exoplatform.com-xwiki-bin-view-ECM-Creating+a+new+Taxonomy+Tree]

- From DMS 2-5, we remove old taxonomy and create a taxonomy tree named **System**. This tree is defined in **dms-taxonomies-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms/** and is automatically created when tomcat runs

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>

  <component>
   <key>org.exoplatform.services.cms.taxonomy.TaxonomyService</key>
   <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyServiceImpl</type>
 </component>

 <external-component-plugins>
    <target-component>org.exoplatform.services.cms.taxonomy.TaxonomyService</target-component>
    <component-plugin>
     <name>predefinedTaxonomyPlugin</name>
      <set-method>addTaxonomyPlugin</set-method>
      <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyPlugin</type>
      <init-params>
       <value-param>
          <name>autoCreateInNewRepository</name>
          <value>true</value>
        </value-param>
        <value-param>
         <name>repository</name>
         <value>repository</value>
        </value-param>
        <value-param>
         <name>treeName</name>
         <value>System</value>
        </value-param>
         <object-param>
           <name>permission.configuration</name>
           <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig">
             <field name="taxonomies">
               <collection type="java.util.ArrayList">
                 <value>
                   <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Taxonomy">
                     <field name="permissions">
                <collection type="java.util.ArrayList">
                   <value>
                     <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission">
                        <field name="identity"><string>any</string></field>
                        <field name="read"><string>true</string></field>
                        <field name="addNode"><string>false</string></field>
                        <field name="setProperty"><string>false</string></field>
                        <field name="remove"><string>false</string></field>
```

```
                      </object>
                    </value>
                    <value>
                      <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission">
                       <field name="identity"><string>*:/platform/administrators</string></field>
                         <field name="read"><string>true</string></field>
                         <field name="addNode"><string>true</string></field>
                         <field name="setProperty"><string>true</string></field>
                         <field name="remove"><string>true</string></field>
                      </object>
                    </value>
                  </collection>
                </field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </object-param>
        <object-param>
          <name>predefined.actions</name>
          <description>description</description>
          <object type="org.exoplatform.services.cms.actions.impl.ActionConfig">
            <field  name="actions">
              <collection type="java.util.ArrayList">
                <value>
                  <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$TaxonomyAction">
                    <field  name="type"><string>exo:taxonomyAction</string></field>
                    <field  name="name"><string>taxonomyAction</string></field>
                    <field  name="description"><string></string></field>
                    <field  name="homePath"><string>dms-system:/exo:ecm/exo:taxonomyTrees/storage/System</stri
                    <field  name="targetWspace"><string>collaboration</string></field>
                    <field  name="targetPath"><string>/Documents</string></field>
                    <field  name="lifecyclePhase"><string>read</string></field>
                    <field  name="roles"><string>*:/platform/administrators</string></field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </object-param>
      <object-param>
...
```

See complete file ^dms-taxonomies-configuration.xml

8.23. Filter all action in File Explorer, ECM Admin

- In DMS 2-5 portlet **ecm** is splited to 4 sub-projects:

1. **core/portlet/ecm/core/main**: Only the java part of the old **core/portlet/ecm** project. The pom of this project creates a jar that will be added in the directory **ecm.war/WEB-INF/lib**
2. **core/portlet/ecm/core/web**: Only the web part of the old core/portlet/ecm project. The pom of this project creates a war that will be the **ecm.war**
3. **core/portlet/ecm/ext/main**: Only the java part of the current extensions. The pom of this project creates a jar that will be added in the directory **ecm.war/WEB-INF/lib**
4. **core/portlet/ecm/ext/config**: Only the configuration of those extensions. The pom of this project creates a jar that will be added in the directory **tomcat/lib**

- Now all action in File Explorer and ECM Admin are filtered for each view. All actions are registered and

manage by **org.exoplatform.webui.ext.UIExtensionManager**. We configurate for actions in **dms-ext-configuration.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/dms/**

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>
  <component>
  <key>org.exoplatform.webui.ext.UIExtensionManager</key>
  <type>org.exoplatform.webui.ext.impl.UIExtensionManagerImpl</type>
  <component-plugins>
   <component-plugin>
    <name>Add Actions</name>
    <set-method>registerUIExtensionPlugin</set-method>
    <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
    <init-params>
     <object-param>
      <name>AddFolder</name>
      <object type="org.exoplatform.webui.ext.UIExtension">
       <field name="type"><string>${typeOfAction}</string></field>
       <field name="name"><string>${ActionName}</string></field>
       <field name="component"><string>${ActionHandle}</string></field>
      </object>
...
```

See complete file "dms-system-configuration.xml"
 Please refer to http://wiki.exoplatform.com/xwiki/bin/view/ECM/UI+Extension+Framework

1 Gadget for last edited documents **LastEditedDocuments** gadget displays last 5 edited documents with document name and edited time. When user click on the document, its content will be displayed in File Explorer

- In **GetEditedDocumentRESTService.class** (exo.ecm.dms.core.component.cms-2.4.jar) we implement Rest Service to get these document.
- Add new file **LastEditedDocuments.xml** in **src/main/webapp/gadgets** for displaying gadget

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
 <ModulePrefs author="eXoPlatform"
  title="Last edited documents"
  directory_title="Last edited documents"
  title_url="http://www.exoplatform.org"
  description="The last edited documents"
  height="230">
 </ModulePrefs>
 <Content type="html">
 <![CDATA[

  <script type="text/javascript" src="/eXoDMSGadgets/javascript/script.js"></script>

  <style type="text/css">
   body{
    font: 11px Tahoma,Verdana,Arial,Helvetica,sans-serif;
    height:100%;
    margin:0 auto;
    padding:0;
   }

   a {
    text-decoration: none;
   }

   ...
   </style>
    <script type="text/javascript">
   function LastEditedDocument() {

   }
```

```
 var siteUrl = 'http://localhost:8080/portal/rest/presentation/document/edit/repository?showItems=5';
 var accessUrl = 'http://localhost:8080/portal/private/classic/contentmanagement/fileexplorer?portal:component

 LastEditedDocument.prototype.makeRequest = function(url, callback) {
  var params = {};
  params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
  params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.JSON;
  var ts = new Date().getTime();
   var sep = "?";
   var refreshInterval = 60;
   if (refreshInterval && refreshInterval > 0) {
     ts = Math.floor(ts / (refreshInterval * 1000));
   }
   if (url.indexOf("?") > -1) {
     sep = "&";
   }
   url = [ url, sep, "nocache=", ts ].join("");
  gadgets.io.makeRequest(url, callback, params);
 }

 LastEditedDocument.prototype.init = function() {
  lastEditedDocument.makeRequest(siteUrl,lastEditedDocument.displayValue);
 }

 LastEditedDocument.prototype.displayValue = function(resp) {
  var data = resp.data.lstDocNode;
  var editedDocList = _gel("ListEditDocumentNode");
  var displayTime = new DisplayTime();
  for (var i = 0; i < data.length; i++) {
   var doc = data[i];
   var AppClass = document.createElement('div');
   var showHTML = "<a target='_top' href='";
   showHTML+= accessUrl;
   showHTML+= "&repository=repository";
   showHTML+= "&drive=";
   showHTML+= doc.driveName;
   showHTML+= "&path=";
   showHTML+= doc.path;
   showHTML+= "'>";
   showHTML+=doc.name;
   showHTML+="</a>";
   showHTML+="<div class='DateTime'>";
   showHTML+="";
   showHTML+=displayTime.timeToPrettyString(doc.dateEdited);
   showHTML+="</div>";
   AppClass.innerHTML = showHTML;
   AppClass.className = 'ItemIcon';
   editedDocList.appendChild(AppClass);
  }
  var title = _gel("BgTitle");
  if (data && (data.length > 0) && title) {
   editedDocList.parentNode.removeChild(title);
  }
  gadgets.window.adjustHeight();
 }

 var lastEditedDocument = new LastEditedDocument();
 gadgets.util.registerOnLoadHandler(lastEditedDocument.init);
</script>

<body>

 <div class="ContentGadGet">
  <div class="BgLeftTitleContent">
   <div class="BgRightTitleContent">
    <div class="BgCentertTitleContent">
     <div class="IconArrow"><span></span></div>
     <div class="TextTitle">Last Edited Documents</div>
     <div class="ClearRight"><span></span></div>
    </div>
   </div>
  </div>
  <div class="BgRightContent">
   <div class="BgLeftContent">
    <div class="BgCenterContent">
     <div id="BgTitle" class="BgTitle">
      <div class="NotAvailableItemIcon">
```

```
        Not available document!
       </div>
      </div>
      <div id="ListEditDocumentNode" class="ListEditDocumentNode">
      </div>
     </div>
    </div>
   </div>
   <div class="BgLeftFooterContent">
    <div class="BgRightFooterContent">
     <div class="BgCenterFooterContent"><span></span></div>
    </div>
   </div>
  </div>
 </body>
]]>
</Content>
</Module>
```

8.24. Set up size of uploading file

- When we upload a file in the File Explorer, we need to limit the size of the file. Maximum size of file is set up in **portlet-preferences.xml** at the location **dms/core/web/portal/.../WEB-INF/conf/portal/group/platform/users**. In this example, we set a default of 30M for the file upload:

```
<portlet-preferences>
...
 <preference>
  <name>uploadFileSizeLimitMB</name>
  <value>30</value>
  <read-only>false</read-only>
 </preference>
...
</portlet-preferences>
```

8.25. Changes in SVN and exobuild since DMS 2-3

Since version 2.3 we changed the structure of the SVN directory.

8.26. The old structure and ECM no more located in this

http://svn.exoplatform.org:3840/projects/ecm/

* Tags location: ~~svn.exoplatform.org:3840/projects/ecm/tags~~ * ~~Trunk location:~~ svn.exoplatform.org:3840/projects/ecm/trunk- * Branches location: ~~svn.exoplatform.org:3840/projects/ecm/branches~~

You still can make a build with old version of ECM (from 2.0.x to 2.2.x) by command :

```
exobuild -product=ecm -exclude=all -build -deploy --version=x.x
```

But you have to change a little in files product.js and module.js of each ECM old version.

For example, in **product.js** file, you have to change

```
product.codeRepo = "ecm/tags/2.1.3" ;
var ecm = Module.GetModule("ecm/tags/2.1.3", {kernel : kernel, core : core, ws : ws,
 eXoPortletContainer : eXoPortletContainer, eXoJcr : eXoJcr, portal : portal});
```

By

```
product.codeRepo = "ecm/dms/tags/2.1.3" ;
var ecm = Module.GetModule("ecm/dms/tags/2.1.3", {kernel : kernel, core : core, ws : ws,
eXoPortletContainer : eXoPortletContainer, eXoJcr : eXoJcr, portal : portal});
```

In **module.js** file, you have to change

```
module.relativeSRCRepo =  "ecm/tags/2.1.3" ;
```

By

```
module.relativeSRCRepo =  "ecm/dms/tags/2.1.3" ;
```

1 The new structure Since 2.3, ECM has been renamed to DMS to better focus on document management.

http://svn.exoplatform.org:3840/projects/ecm/dms

* Tags location: http://svn.exoplatform.org:3840/projects/ecm/dms/tags * Trunk location: http://svn.exoplatform.org:3840/projects/ecm/dms/trunk * Branches location: http://svn.exoplatform.org:3840/projects/ecm/dms/branches

You can make a build with DMS by command

```
exobuild --product=dms --exclude=all --update --build --deploy
```

8.27. New Workflow product has been created for better modularity and flexiblity

There are plans to add more features to workflow and you can see or checkout it at the location

http://svn.exoplatform.org:3840/projects/ecm/workflow * Trunk location: http://svn.exoplatform.org:3840/projects/ecm/workflow/trunk

And we have a new choice to use

```
exobuild -product=workflow -update -build -deploy
```

1 Moved WCM product to the new structure

* The old one: ~~-svn.exoplatform.org:3840/projects/wcm~~ * The new one: http://svn.exoplatform.org:3840/projects/ecm/wcm--

And the same problem in ECM. You have to change two files **product.js** and **module.js** to make wcm build well by command

```
exobuild --product=wcm --update --build --deploy
```

8.28. Added the content-validation which to be used to validate content

Before, this function was integrated in ECM and we can not exclude it with ECM. Since DMS 2.3, this function is optional and we can say yes or no with content validation. We stored the contentvalidation under ecm structure

http://svn.exoplatform.org:3840/projects/ecm/contentvalidation/trunk

Easy to enable the contentvalidation by added parameter "enable-workflow" to build command

```
exobuild -product=dms -update -build -deploy --enable-workflow
```

> **Note**
>
> Finally, you can refer to the [Building from sources|http://wiki.exoplatform.org/xwiki/bin/view/Main/Building+from+sources] to see more instructions.

8.29. Configuration of Publication Awareness of the Content Browser

Before DMS 2.3 all documents of the chosen folder were always shown when using the Content Browser. Now, you can choose to show only published documents.

8.30. UI way

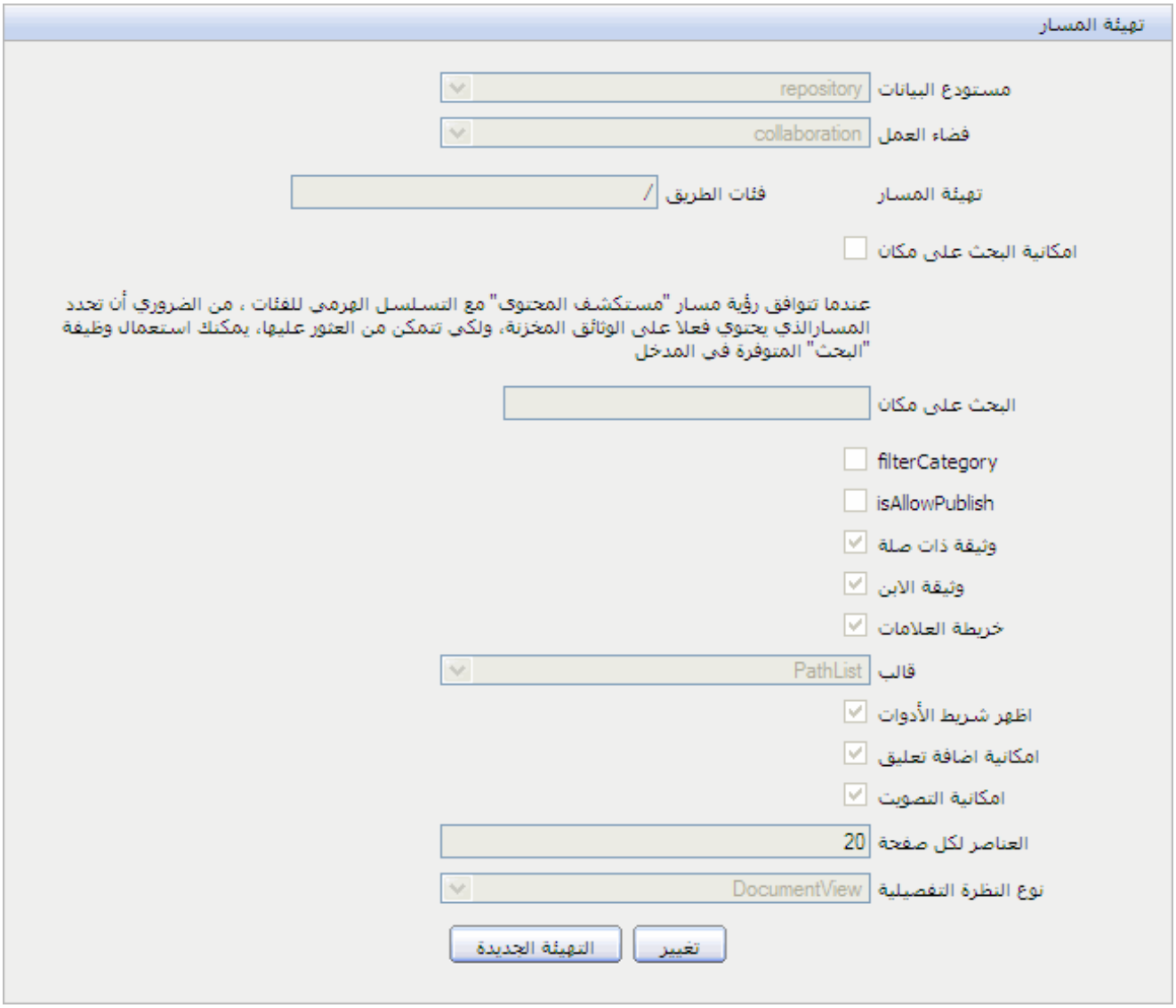
- Click

 icon on the right corner of the Content Browser portlet and select 'Edit' function as :

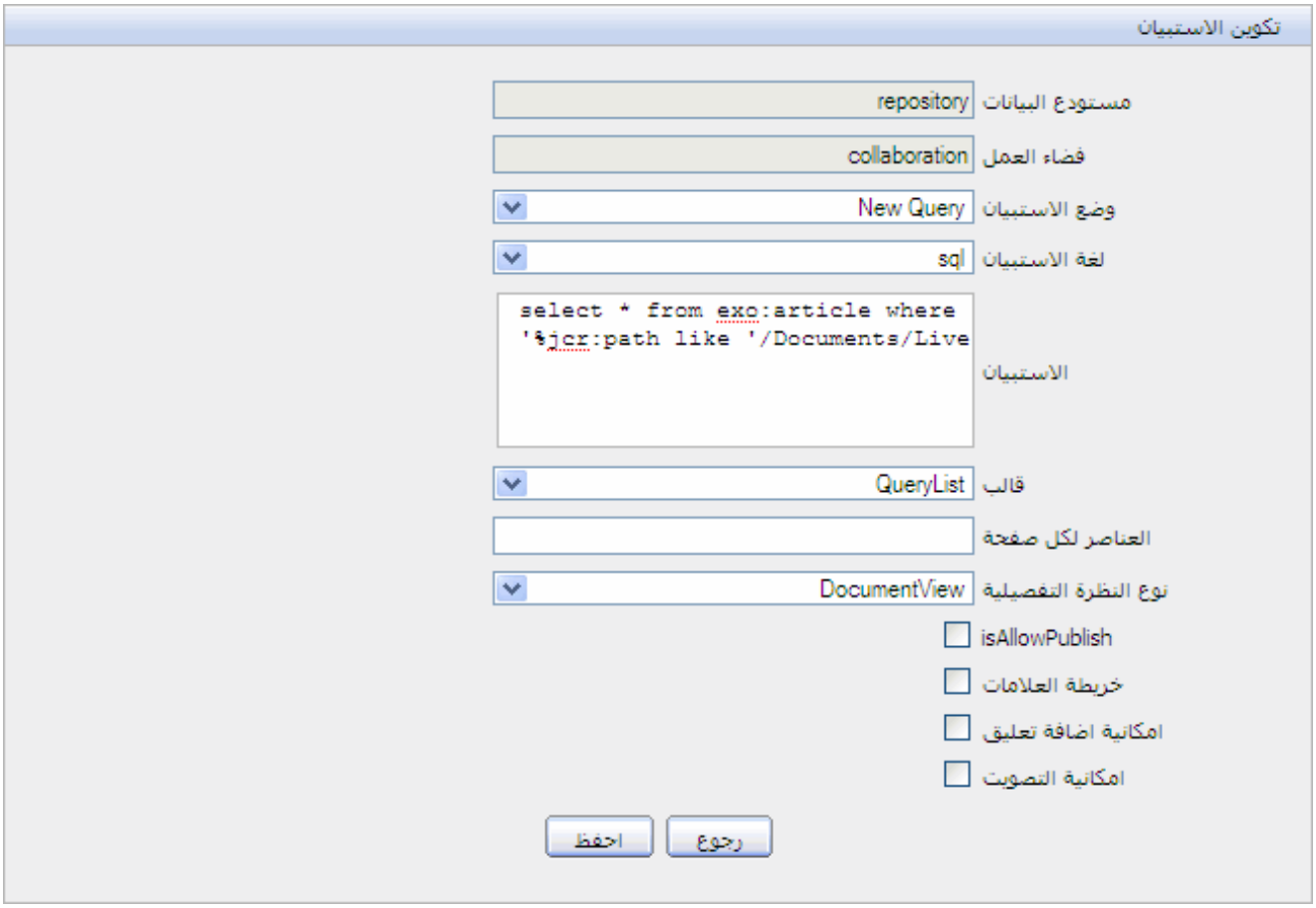
8.30.1. Path config

Select the 'Allow Publish' check box.

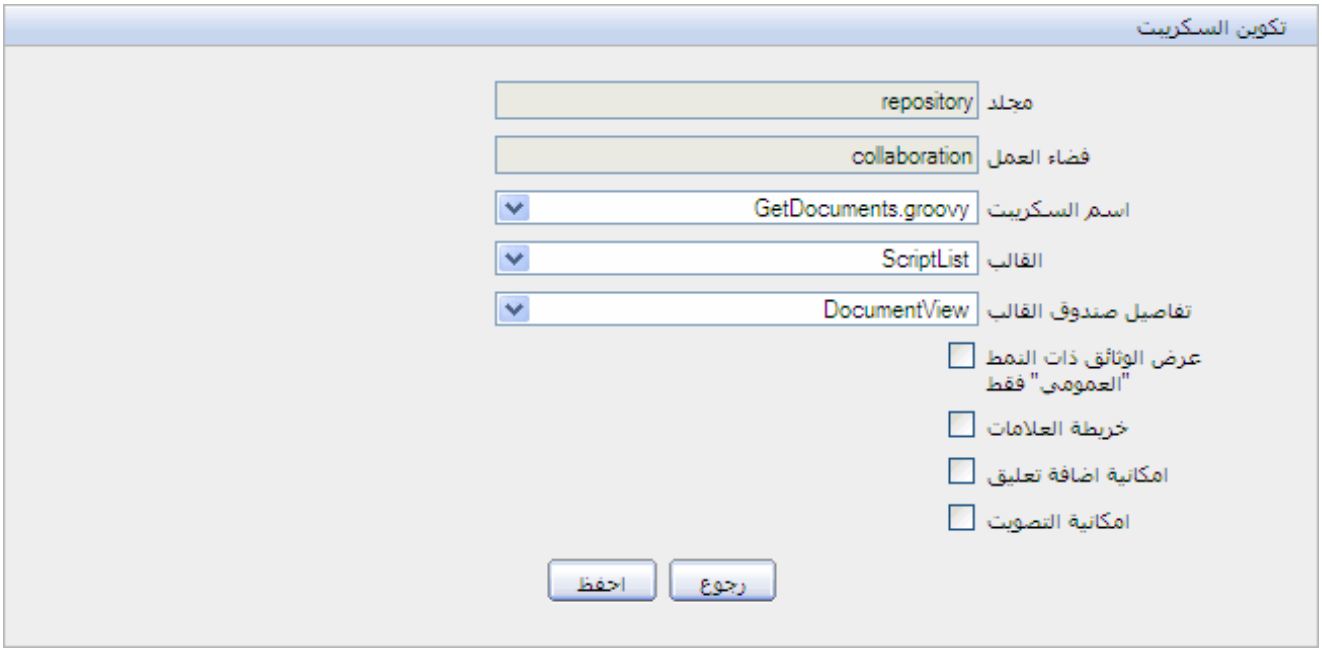
- If you select this check box only **published documents** will be shown.
- Otherwise **all documents** are shown.



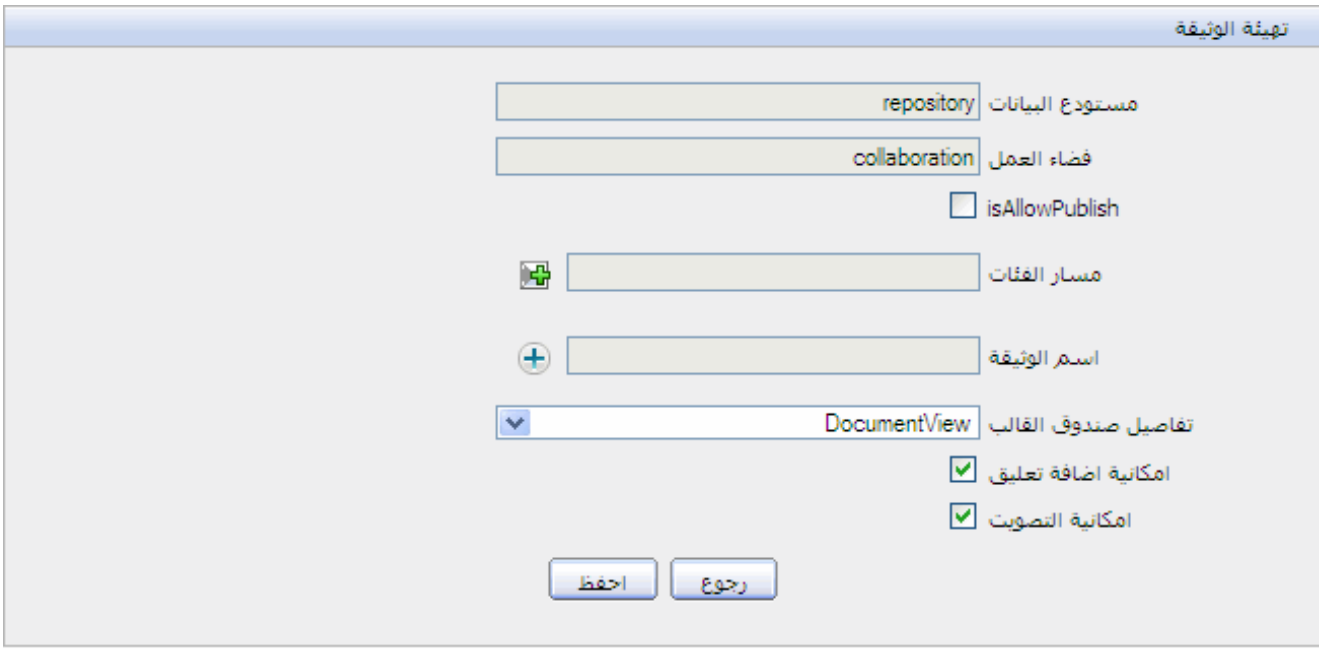
8.30.2. Using query



8.30.3. Using script



8.30.4. Document



8.31. Technical way

In the portlet.xml and portal/group/platform/user/portlet-preferences.xml, insert the new preference value:

```
<preference>
   <name>isAllowPublish</name>
   <value>false</value>
   <read-only>false</read-only>
</preference>
```

8.32. Taxonomy Migration from 2.2.x to 2.5

8.33. Create a Groovy Script

We need to create an action script in order to process the taxonomy migration
 Go to ECMAmin portlet > Advanced configuration > Manage Scripts > Click on the add button.
 Copy the content below into the field "Script contents" and enter "MigrateTaxonomiesAction.groovy" in the field "Script name". Save the script action.

```
/*
 * Copyright (C) 2003-2008 eXo Platform SAS.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License
 * as published by the Free Software Foundation; either version 3
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see<http://www.gnu.org/licenses/>.
 */

import java.util.ArrayList;
```

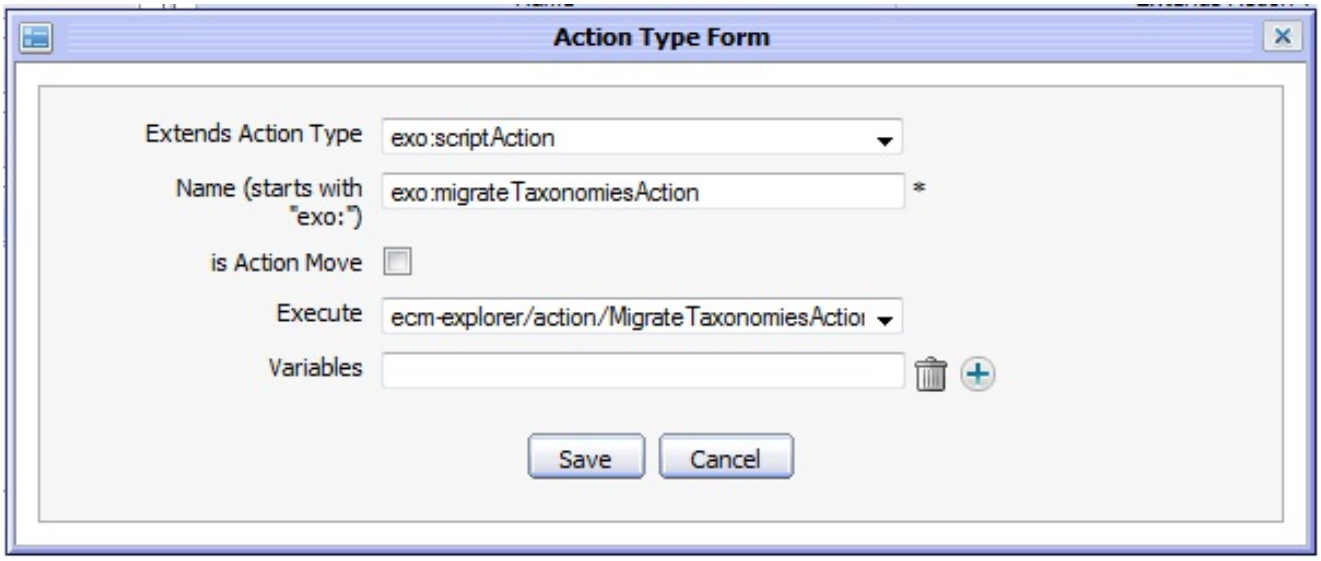
```
import java.util.List;
import java.util.Map;

import javax.jcr.Node;
import javax.jcr.Property;
import javax.jcr.Session;
import javax.jcr.Value;
import javax.jcr.NodeIterator;

import org.apache.commons.logging.Log;
import org.exoplatform.services.cms.scripts.CmsScript;
import org.exoplatform.services.jcr.RepositoryService;
import org.exoplatform.services.jcr.core.ManageableRepository;
import org.exoplatform.services.jcr.ext.app.SessionProviderService;
import org.exoplatform.services.cms.taxonomy.TaxonomyService;
import org.exoplatform.services.log.ExoLogger;

/**
* This script will be used to migrate taxonomies
*/
public class MigrateTaxonomiesAction implements CmsScript {

  private RepositoryService repositoryService_;
  private SessionProviderService seProviderService_;
        private TaxonomyService taxonomyService;
  private static final Log LOG  = ExoLogger.getLogger("ScriptAction.MigrateTaxonomies");

  public MigrateTaxonomiesAction(RepositoryService repositoryService,
      SessionProviderService sessionProviderService, TaxonomyService taxonomyService) {
    this.repositoryService_ = repositoryService;
    seProviderService_ = sessionProviderService;
                this.taxonomyService = taxonomyService;
  }

  public List<String> getPropertyValue(Property prop) throws Exception {
    List<String> uuids = new ArrayList<String>();
    if(prop.getDefinition() != null && prop.getDefinition().isMultiple()) {
      Value[] values =  prop.getValues();
      for (Value value : values) {
        if(value.getString().length() > 0) uuids.add(value.getString());
      }
    }
    return uuids;
  }

  public void execute(Object context) throws Exception {
    Map variables = (Map) context;
    String nodePath = (String)variables.get("nodePath") ;
    String repository = (String)variables.get("repository");
    String srcWorkspace = (String)variables.get("srcWorkspace");
    ManageableRepository manageableRepository = repositoryService_.getRepository(repository);
    Session session = seProviderService_.getSystemSessionProvider(null).getSession(srcWorkspace, manageableRepos
    Node node = (Node)session.getItem(nodePath);
    processMigrateDatas(node, manageableRepository);
    session.save();
    session.logout();
  }

  private void processMigrateDatas(Node node, ManageableRepository manageableRepository) throws Exception {
    if(node.hasProperty("exo:category")) {
     Property category = node.getProperty("exo:category");
     List<String> uuids = getPropertyValue(category);
     Session sessionByWs = null;
     Node categoryNode = null;
     Node linkNode = null;
     for(String workspace : manageableRepository.getWorkspaceNames()) {
       sessionByWs = seProviderService_.getSystemSessionProvider(null).getSession(workspace, manageableRepositor
       for(String uuid : uuids) {
        try {
          categoryNode = sessionByWs.getNodeByUUID(uuid);
          try {
            LOG.info("Migrating the node '"+node.getPath()+"' with category '"+categoryNode.getPath()+"' ")
            taxonomyService.addCategory(node, "System", categoryNode.getPath());
          } catch(Exception e) {
            LOG.error("MIGRATE FAILED! Cannot migrate for node '" +node.getPath()+ "'", e);
          }
        } catch(Exception ne) {
          continue;
```

```
        }
      }

      sessionByWs.save();
      sessionByWs.logout();
    }
    node.removeMixin("exo:categorized");
  } else if(node.hasNodes()) {
    NodeIterator nodeIter = node.getNodes();
    while(nodeIter.hasNext()) {
      processMigrateDatas(nodeIter.nextNode(), manageableRepository);
    }
  } else {
    return;
  }
 }

 public void setParams(String[] params) {}

}
```

Download this file

8.34. Create an Action Type

Go to ECMAmin portlet > Advanced configuration > Create an Action Type > Click on the add button.
 The Action Type Form is shown. Enter the name *exo:migrateTaxonomiesAction_ and choose the* MigrateTaxonomiesAction.groovy *action script in the execute drop down list.
*



8.35. Add a Template for the Action Type

Go to ECMAdmin portlet > Content Presentation > Manage templates > Click on the add button.
 The template form is shown.

 Choose the dialog tab an copy the code shown underneath into the diaglog field.

```

```

```
<%
        def repository = uicomponent.getRepositoryName();
        String[] repositoryField = ["jcrPath=/node/exo:repository", "visible=false", repository] ;
        uicomponent.addHiddenField("repository", repositoryField) ;
%>
<div class="UIFormWithTitle">
  <div class="TitleBar"><%=_ctx.appRes(uicomponent.getId() + ".title")%></div>
        <% uiform.begin() %>
  <div class="HorizontalLayout">
                <table class="UIFormGrid">
                  <tr>
              <td class="FieldLabel"><%=_ctx.appRes("ScriptAction.dialog.label.id")%></td>
              <td class="FieldComponent">
                      <%
                        String[] fieldId = ["jcrPath=/node", "mixintype=exo:move", "editable=false", "visible=if
                        uicomponent.addMixinField("id", fieldId) ;
                      %>
                    </td>
                  </tr>
                        <tr>
                          <td class="FieldLabel"><%=_ctx.appRes("ScriptAction.dialog.label.name")%></td>
                          <td class="FieldComponent">
                                  <%
                                                String[] fieldName = ["jcrPath=/node/exo:name", "validate=empty"
                                                uicomponent.addTextField("actionName", fieldName);
                                  %>
                                </td>
                        </tr>
                        <tr>
                          <td class="FieldLabel"><%=_ctx.appRes("ScriptAction.dialog.label.lifecycle")%></td>
                          <td class="FieldComponent">
                                  <%
                                                String[] fieldLifecycle = ["jcrPath=/node/exo:lifecyclePhase", "
                                                uicomponent.addSelectBoxField("lifecycle", fieldLifecycle)
                                  %>
                                </td>
                        </tr>
                  <tr>
                                <td class="FieldLabel">Roles</td>
                                <td class="FieldComponent">
                                        <%
                                                String[] fieldRoles = ["jcrPath=/node/exo:roles", "selectorActio
                                                uicomponent.addActionField("roles", fieldRoles) ;
                                        %>
                                </td>
                        </tr>

                 </table>
                <%/* start render action*/%>
                <div class="UIAction">
                        <table align="center" class="ActionContainer">
                                <tr>
                                        <td align="center">
                                          <% for(action in uicomponent.getActions()) {
                                                        String actionLabel = _ctx.appRes(uicomponent.getName(
                                                              String link = uicomponent.event(action) ;
                                          %>
                                                  <a href="$link" class="ActionButton LightBlueStyle">
                                                               <div class="ButtonLeft">
                                                                 <div class="ButtonRight">
                                                                         <div class="ButtonMiddle">
                                                                                      $actionLabel
                                                                   </div>
                                                                 </div>
                                                        </div>
                                                  </a>
                                          <%}%>
                                </td>
                          </tr>
                        </table>
                </div>
                <%/* end render action*/%>
  </div>
  <%uiform.end()%>
</div>
```

Download this file

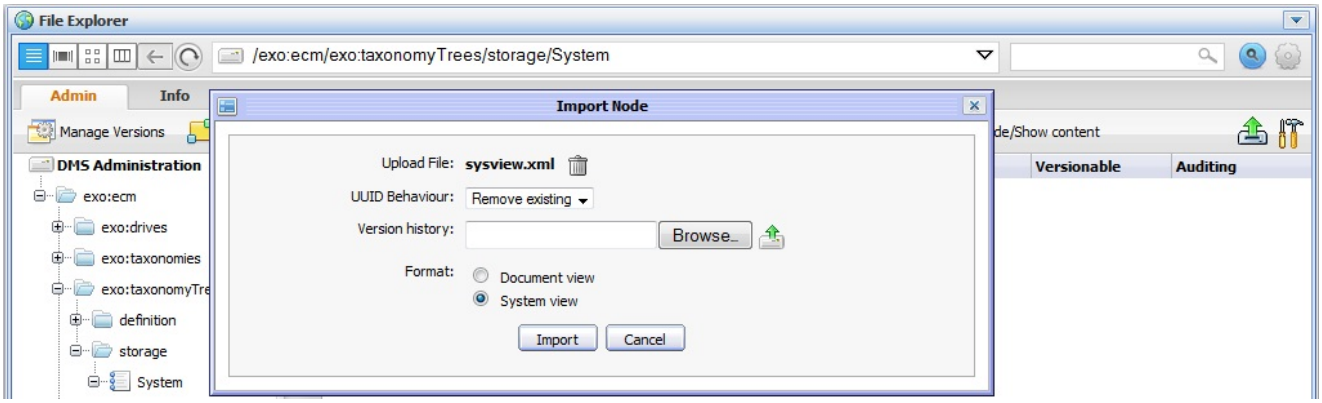
You can leave the view field with content is "test" or anything you want. Click save button to commit the changes.

8.36. Replace new BaseActionLauncherListener class

Download this file to replace the old BaseActionLaucherListener class

8.37. Import Data with References to the Imported Taxonomies

- Import all taxonomies which you exported from an old version of DMS into dms-system:/exo:ecm/exo:taxonomyTrees/storage/System. Make sure you don't have any same name taxonomy node in this tree. You can do it easy by remove the default taxonomies(calendar, cms) before do the import action. In the import form, you have to choose the UUID behavior is "Remove existing" to ensure the category is the same as before.



- In the collaboration workspace create a folder.(For ex: "Migrate datas").
- Add an action for this folder using the action type "exo:migrateTaxonomiesAction" and the lifecycle "Add".
- Import all data with references to imported taxonomies into the folder "Migrate datas". In this step, you can choose the UUID behavior is "Create new"

Chapter 9. Development

9.1. Publication Service

9.2. Background

The previous publication model in eXo ECM is based on moves of contents from specific JCR locations to other ones. More precisely, content is dropped by the user to a given location. This has the effect to start a workflow instance. After the workflow has received the approvals from appropriate users, it moves the content to a live location and then to a backup location. However this current situation has several limitations:

- The workflow is at the centre of the publication process, so implementing alternate publication flavours is difficult. In addition, monitor and control is tedious,
- The publication states of contents is represented by the possible JCR locations: "Validation request", "Pending", "Published", "Backed up", "Trash". It is not possible to adapt easily this set,
- In some situations, it is preferable to have the content to stay at the initial location instead of being moved,
- We don't have a centralized UI approach in JCR File Explorer to control publication.

This document specifies how the Publication will be improved in eXo ECM. In particular, it is driven by the need of the CG95 customer project. When the implementation is over, we will be virtually able to implement any publication lifecycle requested by customer projects.

9.3. Design

9.3.1. Introduction

At a high level, a new "Publication" service will be created. This service will be invoked by all entities that need to retrieve or specify publication information. For example : User interfaces, workflow instances, ECM actions. It will accept plugins that specify a publication lifecycle model. By default, eXo ECM will provide two types of plugins :

- WorkflowPublicationPlugin. A Workflow instance is triggered and will contribute to liven up the lifecycle. The contents are moved from one JCR location to another one.
- StaticAndDirectPublicationPlugin. The contents are directly pushed in publication by the user action. They stay at the original location. The requirements of this plugin are those of the CG95 project.

When a customer project comes with non-standard publication lifecycle requirements, it will be easy for us to create a new plugin. Finally, a new mixin type will be created in all cases to hold common publication information. Each plugin will be able to extend it and add its own specific properties.

9.3.2. Publication Service

This service will be added to the main ECM component project.

When eXo is initialized, publication plugins will be added to the service. They will define lifecycles, which modelize states, transitions and actions. The addPublicationPlugin() method is used towards this goal. As for the method getPublicationPlugins(), it lists all plugins which have been registered so far. That way, it is possible to retrieve the names of all possible lifecycles. The idea is that each JCR node first needs to be enrolled in a publication lifecycle in order to be published, thanks to enrollNodeInLifecycle(). In its parameters, this method expects a name of a lifecycle, which corresponds to a publication plugin name. The isNodeEnrolledInLifecycle() checks whether a specified node has been added in a lifecycle.

When a node is enrolled, a mixin type is added to it. It specifies the current state in the lifecycle. The method changeState() allows to update the current state of the specified node. It will delegate to the plugin of the used lifecycle. An atypical point in that service is that it allows generating a WebUI form corresponding to a state in the lifecyle. This is done through the getStateUI() method. The main intent for that is to enable the JCR File Explorer to display a specific panel and allow the user specifying information for the current state. The returned WebUI form will possibly show fields where user can specify information and push buttons.

Finally, there exists the getUserInfo(), getStateImage() and getLog() methods to respectively retrieve textual description, visual description and history information. The snippet below defines its interface. The provided Javadoc needs to be added to the source code when it is implemented.

```
package org.exoplatform.services.ecm.publication;

import java.util.HashMap;
import java.util.Locale;
import java.util.Map;

import javax.jcr.Node;

public interface PublicationService {

  /**
   * Add a Publication Plugin to the service.
   * The method caches all added plugins.
   *
   * @param p the plugin to add
   */
  public void addPublicationPlugin(PublicationPlugin p);

  /**
   * Retrieves all added publication plugins.
   * This method is notably used to enumerate possible lifecycles.
   *
   * @return the added publication plugins
   */
  public Map<String,PublicationPlugin> getPublicationPlugins();


  /**
   * Update the state of the specified node.
   * This method first inspects the publication mixin bound to the specified
   * Node. From that mixin, it retrieves the lifecycle registered with the
   * node. Finally, it delegates the call to the method with same name in the
   * plugin that implements the lifecycle.
   *
   * @param node the Node whose state needs to be changed
   * @param newState the new state.
   * @param context a Hashmap containing contextual information needed
   * to change the state. The information set is defined on a State basis.
   * A typical example is information submitted by the user in a user
   * interface.
   *
   * @throws NotInPublicationLifecycleException in case the Node has not
   * been registered in any lifecycle yet (in other words, if no publication
   * mixin has been found).
   * @throws IncorrectStateUpdateLifecycleException if the update is not
   * allowed
   * @throws Exception the exception
```

```
     */
    public void changeState(Node node, String newState, HashMap<String, String> context)
    throws NotInPublicationLifecycleException, IncorrectStateUpdateLifecycleException, Exception;

    /**
     * Retrieves an image showing the lifecycle state of the specified Node.
     * The method first inspects the specified Node. If it does not contain
     * a publication mixin, then it throws a NotInPublicationLifecycleException
     * exception. Else, it retrieves the lifecycle name from the mixin,
     * selects the appropriate publication plugin and delegates the call to it.
     *
     * @param node the node from which the image should be obtained
     * @param locale the locale
     *
     * @return an array of bytes corresponding to the image to be shown to the
     * user
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public byte[] getStateImage(Node node,Locale locale)throws NotInPublicationLifecycleException ,Exception;

    /**
     * Retrieves the name of the publication state corresponding to the
     * specified Node.
     * This method first inspects the specified Node. If it does not contain
     * a publication mixin, then it throws a NotInPublicationLifecycleException
     * exception. Else, it retrieves the current state name from the mixin.
     * Possible examples of State names are : "draft", "validation requested",
     * "publication pending", "published", "backed up", "validation refused".
     *
     * @param node the node from which the publication state should be retrieved
     *
     * @return a String giving the current state.
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public String getCurrentState(Node node) throws NotInPublicationLifecycleException ,Exception;

    /**
     * Retrieves description information explaining to the user the current
     *
     * This method first inspects the specified Node. If it does not contain
     * a publication mixin, then it throws a NotInPublicationLifecycleException
     * exception. Else, it retrieves the lifecycle name from the mixin,
     * selects the appropriate publication plugin and delegates the call to it.

     *
     * @param node the Node from which user information should be retrieved
     * @param locale the locale
     *
     * @return a text message describing the state of the current message.
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public String getUserInfo(Node node, Locale locale) throws NotInPublicationLifecycleException ,Exception;

    /**
     * Retrieves the history of publication changes made to the specified Node.
     *
     * This method first inspects the specified Node. If it does not contain
     * a publication mixin, then it throws a NotInPublicationLifecycleException
     * exception. Else, it retrieves the lifecycle name from the mixin,
     * selects the appropriate publication plugin and delegates the call to it.
     *
     * Log entries are specified as a multi-valued property of the publication
     * mixin.
     *
     * @param node the Node from which the history Log should be retrieved
     *
     * @return a String array with 2 dimensions. The first dimension contains
```

```
     * each log entry. The second dimension contains each information in a log
     * entry, which are : date, name of the new state, involved user, additional
     * information.
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public String[][] getLog(Node node) throws NotInPublicationLifecycleException, Exception;

    /**
     * Adds the log.
     *
     * @param node the node
     * @param log the log
     *
     * @throws NotInPublicationLifecycleException the not in publication lifecycle exception
     * @throws Exception the exception
     */


    /**
     * Adds a log entry to the specified Node.
     * The specified array of String defines the Log information to be added.
     * Log entries are specified as a multi-valued property of the publication
     * mixin.
     *
     * @param node the Node from which the history Log should be updated
     * @param log the Log information to be added
     * log contains : date, newState, userInvolved, key for additionalInformation in locale with possible subsitut
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public void addLog(Node node, String[] log) throws NotInPublicationLifecycleException, Exception;

    /**
     * Determines whether the specified Node has been enrolled into a
     * lifecycle.
     *
     * @param node the Node from which the enrollment should be evaluated
     *
     * @return true of the Node is enrolled
     *
     * @throws Exception the exception
     */
    public boolean isNodeEnrolledInLifecycle(Node node) throws Exception;

    /**
     * Retrieves the name of the lifecycle in which the specified Node has
     * been enrolled.
     *
     * @param node the Node from which the enrollment should be retrieved
     *
     * @return the name of the lifecycle corresponding to the specified Node
     *
     * @throws NotInPublicationLifecycleException in case the Node has not
     * been registered in any lifecycle yet (in other words, if no publication
     * mixin has been found).
     * @throws Exception the exception
     */
    public String getNodeLifecycleName(Node node) throws NotInPublicationLifecycleException, Exception;

    /**
     * Retrieves the description of the lifecycle in which the specified Node
     * has been enrolled.
     *
     * This method first inspects the specified Node. If it does not contain

     * a publication mixin, then it throws a NotInPublicationLifecycleException
     * exception. Else, it retrieves the lifecycle name from the mixin,
     * selects the appropriate publication plugin and delegates the call to it.
     *
     * @param node the Node from which the enrollment should be retrieved
     *
     * @return the description of the lifecycle corresponding to the specified
```

```
    * Node
    *
    * @throws NotInPublicationLifecycleException in case the Node has not
    * been registered in any lifecycle yet (in other words, if no publication
    * mixin has been found).
    * @throws Exception the exception
    */
   public String getNodeLifecycleDesc(Node node) throws NotInPublicationLifecycleException ,Exception;

   /**
    * Enroll the specified Node to the specified lifecycle.
    * This method adds a publication mixin to the specified Node. The lifecycle
    * name is the one specified as parameter. By default, the state is set
    * to "enrolled".
    *
    * @param node the Node to be enrolled in the specified lifecycle
    * @param lifecycle the name of the lifecycle in which the Node should be
    * enrolled
    *
    * @throws AlreadyInPublicationLifecycleException the already in publication lifecycle exception
    * @throws Exception the exception
    */
   public void enrollNodeInLifecycle(Node node, String lifecycle) throws AlreadyInPublicationLifecycleException,

   /**
    * Unsubcribe node that in publication lifecyle.
    *
    * @param node the node
    *
    * @throws NotInPublicationLifecycleException the not in publication lifecycle exception
    * @throws Exception the exception
    */
   public void unsubcribeLifecycle(Node node) throws NotInPublicationLifecycleException, Exception;

   /**
    * Get localized log messages and substitute variables.
    *
    * @param locale : the locale to use
    * @param key : the key to translate
    * @param values : array of string to susbtitute in the string
    *
    * @return the localized and substitute log
    *
    * @result a string localized and where values are substitute
    */
   public String getLocalizedAndSubstituteLog(Locale locale, String key, String[] values);

   /**
    * Gets the localized and substitute log for current node.
    * Base on lifecycle that node is enroll. The method call to get message for each lifecycle
    *
    * @param node the node
    * @param locale the locale
    * @param key the key
    * @param values the values
    *
    * @return the localized and substitute log
    *
    * @throws NotInPublicationLifecycleException the not in publication lifecycle exception
    * @throws Exception the exception
    */
   public String getLocalizedAndSubstituteLog(Node node, Locale locale, String key, String[] values) throws NotIn
}
```

9.3.3. Plublication mixin type

1. new "publication:publication" mixin type is used to decorate nodes enrolled in a publication lifecycle. It is defined as follows:

Table 9.1.

Property	Meaning
publication:lifecycleName	Name of the lifecycle.
publication:currentState	Name of the current state in the lifecycle. The default value is "enrolled".
publication:history	An array of Strings. Each String is a log entry. It contains comma "," separated tokens. The tokens are the log information themselves (date, name of the new state, involved user, additional information).

Please node that each property of this mixin type has an "onParentVersion" flag set to "IGNORE". This is to avoid taking into account publication information when making versions.

9.4. Publication plugins

As said earlier, the Publication Service is enriched by many publication plugins. They will implement the lifecycle of the publication model. They will be identified by a name. This name will be specified as a property of the publication mixin. That way, the Publication Service will easily retrieve which publication plugin should be used when delegating a call.

Each plugin is packaged in its own jar file. The jar file contains an XML configuration file that registers the plugin to the Publication Service (using external configuration). Apart from the class of the plugin itself, the jar also contains :

- the possible WebUI classes to define publication state forms including corresponding action listeners.
- images representing the state to the user (typically a graph highlighting the current state).
- resource bundles to specifiy user information to describe the current state.

The fact that plugins are contained and defined in separate jar files will make it easy to add new lifecycles.

The most significant method of the plugins is changeState(), which applies the lifecycle and modifies the state of a Node.

The snippet below defines its interface. The provided Javadoc needs to be added to the source code when it is implemented.

```
package org.exoplatform.services.ecm.publication;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Locale;
import java.util.Map;

import javax.jcr.Node;

import org.exoplatform.container.component.BaseComponentPlugin;
import org.exoplatform.webui.core.UIComponent;
import org.exoplatform.webui.form.UIForm;

/**
 * Base class of Publication plugins.
 * Publication plugins implement a publication lifecycle. Each time a new
 * custom lifecycle needs to be defined, a new plugin has to be implemented
 * and registered with the Publication Service.
```

```
 *
 * The getName() method in the parent class is used to identify the lifecycle.
 * The getDescription() method in the parent class is used to describe the
 * lifecycle. Internationalization resource bundles are used in the
 * implementation of the method.
 */
public abstract class PublicationPlugin extends BaseComponentPlugin {

  /**
   * Retrieves all possible states in the publication lifecycle.
   *
   * @return an array of Strings giving the names of all possible states
   */
  public abstract String[] getPossibleStates();

  /**
   * Change the state of the specified Node.
   * The implementation of this method basically retrieves the current
   * state from the publication mixin of the specified Node. Then, based on
   * the newState, it is able to determine if the update is possible. If
   * yes, appropriate action is made (eg: launch a publication workflow). In
   * all cases, the current state information is updated in the publication
   * mixin of the specified Node.
   *
   * @param node the Node whose state needs to be changed
   * @param newState the new state.
   * @param context a Hashmap containing contextual information needed
   * to change the state. The information set is defined on a State basis.
   *
   * @throws IncorrectStateUpdateLifecycleException if the update is not
   * allowed
   * @throws Exception the exception
   */
  public abstract void changeState(Node node,
      String newState,
      HashMap<String, String> context)
  throws IncorrectStateUpdateLifecycleException, Exception;

  /**
   * Retrieves the WebUI form corresponding to the current state of the
   * specified node.
   * There are two cases here. Either the form contains read only fields (when
   * the state is supposed to be processed by an external entity such as a
   * Workflow). Or the form has editable fields or buttons (in the case the
   * user can interfere. In that case, some action listeners are leveraged.).
   * In all cases, all UI and listener classes are provided in the JAR
   * corresponding to the PublicationPlugin.
   * The method first inspects the specified Node. If it does not contain
   * a publication mixin, then it throws a NotInPublicationLifecycleException
   * exception. Else, it retrieves the lifecycle name from the mixin,
   * selects the appropriate publication plugin and delegates the call to it.
   *
   * @param node the Node from which the state UI should be retrieved
   * @param component the component
   *
   * @return a WebUI form corresponding to the current state and node.
   *
   * @throws Exception the exception
   */
  public abstract UIForm getStateUI(Node node, UIComponent component) throws Exception;

  /**
   * Retrieves an image showing the lifecycle state of the specified Node.
   * The implementation of this method typically retrieves the current state
   * of the specified Node, then fetches the bytes of an appropriate image
   * found in the jar of the plugin. This image is supposed to be shown in
   * the publication dialog of the JCR File Explorer Portlet.
   *
   * @param node the node from which the image should be obtained
   * @param locale the locale
   *
   * @return an array of bytes corresponding to the image to be shown to the
   * user
   *
   * @throws IOException Signals that an I/O exception has occurred.
   * @throws FileNotFoundException the file not found exception
   * @throws Exception the exception
   */
  public abstract byte[] getStateImage(Node node, Locale locale) throws IOException,FileNotFoundException,Except
```

```
/**
 * Retrieves description information explaining to the user the current
 * publication state of the specified Node. Possible examples are
 * - "The document has been submitted to the following group for validation:
 * /organization/management.".
 * - "The document has been validated and will be published from
 * May 3rd 10:00am to May 3rd 10:00pm. At that time, it will be unpublished
 * and put in a backup state.".
 * - "The document is in draft state. At any time you can turn it to
 * published state."
 *
 * The returned message should be obtained from internationalization
 * resource bundles (ie not hardcoded).
 *
 * @param node the node from which the publication state should be retrieved
 * @param locale the locale
 *
 * @return a String giving the current state.
 *
 * @throws Exception the exception
 */
public abstract String getUserInfo(Node node, Locale locale) throws Exception;

/**
 * Retrieves the lifecycleName.
 *
 * @return a String giving the lifecycleName
 */

public String getLifecycleName() {
  return getName();
}

/**
 * Retrieves the description of the plugin.
 *
 * @param node the node
 *
 * @return a String giving the description
 */
public String getNodeLifecycleDesc(Node node) {
  return getDescription();
}

/**
 * Return if the plugin can add the specific mixin for the publication.
 *
 * @param node the node to add the mixin
 *
 * @return boolean
 *
 * @throws Exception the exception
 */
public abstract boolean canAddMixin (Node node) throws Exception;

/**
 * Add the specific plugin mixin to the node.
 *
 * @param node the node
 *
 * @throws Exception the exception
 */
public abstract void addMixin (Node node) throws Exception;

/**
 * Retrieves a node view of the specific node in a context
 *
 * @param node the node
 * @param context the context
 *
 * @return the node to view
 *
 * @throws Exception the exception
 */
public abstract Node getNodeView(Node node, Map<String,Object> context) throws Exception;

/**
 * Get localized log messages and substitute variables.
```

```
    *
    * @param locale : the locale to use
    * @param key : the key to translate
    * @param values : array of string to susbtitute in the string
    *
    * @return the localized and substitute log
    *
    * @result a string localized and where values are substitute
    */
   public abstract String getLocalizedAndSubstituteMessage(Locale locale, String key, String[] values) throws Exc
}
```

By default, eXo will come with two out-of-box publication plugins. It will be possible to define new plugins if they do not fit the customer project needs.

9.4.1. WorkflowPublicationPlugin

The lifecycle behind this publication plugin is the one existing is eXo ECM so far. A document is first dropped to the "/Documents/validation requests" JCR folder, where an ECM action triggers a validation workflow. This workflow has a validation task assigned to a specific group in the organization. When a relevant user activates this task, he can either validate the document, refuse the document (the user needs to change the document), refuse the request (the validation request is rejected) or delegate to another user. If he accepts the document, then he has to specify start and end publication dates. Before the first date is met, the document is moved to "/Documents/pending". Then it is moved to "/Documents/live". Finally, when the end publication date is met, the document is moved to a backup directory.

Here is how this lifecycle is implemented and translated into the new Publication Plugin mechanism :

Table 9.2.

Method	**Specification**
getName()	Returns "WorkflowPublicationPlugin"
getDescription()	Returns "This publication lifecycle triggers a validation workflow and moves the content to appropriate locations of the JCR. Each location corresponds to a specific state."
getPossibleStates()	"enrolled", "validation requested", "publication pending", "publication refused", "publication rejected", "published", "backed up" Please node that the delegation state is not embodied. In fact, the publication state will still remain "publication pending" in this situation. A log entry will be added to the history log, to keep track of the delegation.
changeState()	* If the current state is "enrolled" and the new state is "validation requested", then update the mixin and triggers a validation workflow. This situation typically arises when a PublicationAction is triggered in the validation requests folder. Update the state information in the mixin. Append a log entry. **If the current state is "validation requested" and the new state is "publication pending", then moves the document from the validation request folder to the pending folder. Update the state information in**

Method	Specification
	the mixin. Append a log entry. ** If the current state is "validation requested" and the new state is "publication rejected", then do nothing. Update the state information in the mixin. Append a log entry. **If the current state is "validation requested" and the new state is "publication rejected", then move the document to the trash folder. Update the state information in the mixin. Append a log entry. ** If the current state is "publication pending" and the new state is "published", then move the document to the live folder. Update the state information in the mixin. Append a log entry. **If the current state is "published" and the new state is "backed up", then move the document to the back up folder. Update the state information in the mixin. Append a log entry. In all other cases, throw an IncorrectStateUpdateLifecycleException to indicate that the state update is not possible.
getStateUI()	For now, return a blank form (we will possibly improve this in the future).
getStateImage()	Return an image showing the possible states in boxes, with possible connections between them. Highlight the box corresponding to the current state.
getUserInfo()	* If the current state is "enrolled", return "This document has been enrolled into this lifecycle. To trigger a validation request, it needs to be manually dropped to the SPECIFY DYNAMICALLY THE VALIDATION REQUEST FOLDER LOCATION folder." **If the current state is "validation requested", return "The document has been requested for validation by user SPECIFY DYNAMICALLY THE USER NAME. Users of group SPECIFY DYNAMICALLY THE VALIDATOR GROUP NAME are now validating the document." ** If the current state is "publication pending", return "The document publication has been accepted. It will be published from SPECIFY DYNAMICALLY THE START PUBLICATION DATE to SPECIFY DYNAMICALLY THE END PUBLICATION DATE. **If the current state is "publication refused", return "The publication has been refused. The document needs to be revised. The originator SPECIFY DYNAMICALLY THE ORIGINATOR NAME should change the content, open the Workflow Portlet and activate the task corresponding to his request. ** If the current state is "publication rejected", return "The publication

Method	Specification
	has been rejected by user SPECIFY DYNAMICALLY THE USER. The document has been moved to SPECIFY DYNAMICALLLY THE TRASH LOCATION. A new document should be created and a new validation request be made. **If the current state is "published", return "The document is published and located at DYNAMICALLY SPECIFY THE LIVE FOLDER. The publication will end at DYNAMICALLY SPECIFY THE END PUBLICATION DATE". ** If the current state is "backed up", return "The document is no more published and is in back up state. It is located at DYNAMICALLYT SPECIFY THE BACKUP FOLDER.

The plugin parameters define the following information:

- JCR locations of validation requests, pending documents, live documents, trashed documents and backed up documents. This means that the validation workflow will no more contain this information. It will delegate all moves to the publication service.
- Organization groups of validators.
- Name of the validation workflow to trigger.

The mixin type "publication:WorkflowPublication" inherits from "publication:publication". It adds the following fields:

Table 9.3.

Property	Meaning
publication:validator	The user which validate the content of node
publication:businessProcess	The process which user working in
publication:backupPath	The path which store node when node is unpublished

Please node that each property of this mixin type has an "onParentVersion" flag set to "IGNORE". This is to avoid taking into account publication information when making versions.

The workflow action listeners should be rewritten. From now, instead of directly orchestrating and updating documents, they will delegate to the publication service.

9.4.2. StaticAndDirectPublicationPlugin

This lifecycle implements the publication flavour chosen by the CG95. In this model, the user directly publishes content. This means that there is no validation. In addition, the content Nodes stay at their original location.

The model presents 3 states only: "enrolled", "non published", "published". The different versions of the Node have different publication states. This means that any version of a Node can be made active and published. Only one version can be in "published" state at a time. The "enrolled" state is specified to comply with the publication model. The "non published" state is on when no version of the Node is published. The "published" state is on when one version of the Node is published. There will be additional properties to the mixin type to specify the publication state of each version.

In addition to the publication state, this lifecycle has a visibility flag, which allows to specify whether a published content can be seen by guest users (ie "public" content) or by private users who can access a Portal. In both case, the plugin defines JCR permissions accordingly.

Here is how the lifecycle is implemented:

Table 9.4.

Method	**Specification**
getName()	Returns "StaticAndDirect"
getDescription()	Returns "This publication lifecycle keeps the content at their original place. Any version of the Node can be published."
getPossibleStates()	enrolled, non published, published
changeState()	* If the new state is "enrolled", directly switch the state to "draft" (this means that there will never be nodes in "enrolled" states). **If the current state is "draft" and the new state is "published", the context parameter should contain the node version UUID to be published and the visibility level. If not, throw an IncorrectStateUpdateLifecycleException. Extract the current state of the version Node to publish. If that Node has no state (not present in the list yet) or is in "Draft" state, make the version Node to be published as "Published" in the "exo:versionsPublicationStates". If not, throw an IncorrectStateUpdateLifecycleException exception. Extract the "visibility" information from the context and update accordingly the "exo:visibility" mixin property. If the visibility is "public", modify the access permission so that the "any" user can at least view the content. If the visibility is "private", make sure that no "any" permission entry is set. To finish, make the version to be published as the active version. Append a log entry (should specify the version). ** Same if the current state is "published" and a new state is "published". This situation arises when a version Node is already published and another Version node needs to be published. In addition, extract a possible previously published Node version from the

Method	Specification
	"exo:versionsPublicationStates" mixin property. Make that version Node state "Backed up". If the current state is "published" and the new state is "unpublished", extract the current published version Node from the "exo:versionsPublicationStates" and make this version as "Backed up". Append a log entry (should specify the version). In all other cases, throw an IncorrectStateUpdateLifecycleException to indicate that the state update is not possible.
getStateUI()	**If the current state is "enrolled" or "non published", display a text stating "No version is currently published. Please select a version to be published". Display a version tree of the current node (including the current version. This tree is the same as the one shown in the versions JCR File Explorer dialog). Allow the user to select one Node. A selector will be labelled "Visibility:" and will have two options "public" and "public". A "Publish" button will be at the bottom. When that button is pressed, invoke changeState() by specifying "published" as newState and the selected version Node plus the selected visibility in the context. ** If the current state is "published", display a text stating "The following version is published: version id - version label.". Ideally, a hypertext link should allow the user to view the document when he clicks it. An "Unpublish" button will allow to unpublish the Node. In that case, the changeState() method is invoked with newState set to "unpublished". Apart from that, the WebUI Form is the same as the one in the previous case. getStateImage() Return an image with two entities : "unpublished" and "published". The "published" entity has multiple boxes, to show that multiple versions can be published. The two entities are connected by a two-arrows link. Depending on the current state, one entity or the other is highlighted.
getUserInfo()	* If the current state is "enrolled" or "non published", return "This document is not currently published". **If the current state is "published", return "The version versionid (labelled versionlabel) is currently published. Its visibility is visibilitylevel. This means that all/restricted users can view the content."**

The mixin type "publication:staticAndDirectPublication" inherits from "publication:publication". It adds the following fields:

Table 9.5.

Property	Meaning
publication:visibility	Defines whether the content is visible to all users ("public") or to a specific set of users who can access a Portal ("private"). In this last case, JCR permissions should be set accordingly.
publication:versionsPublicationStates	The type of this property is an array of String. Each string is a coma separated entry, containing in first position a version node UUID and in second position a version publication state, that is : UUID,State. The possible values for the version states are "draft", "published", "backed up".

9.5. JCR File Explorer Dialog

1. new icon will be made available to the JCR File Explorer toolbar. When the user hits this icon, a new dialog appears. This dialog contains information related to the publication state and lifecycle bound to the current node.

If the current node has not been enrolled in any lifecycle yet (ie no publication mixin has been added), then a popup message appears. It contains the text: "The current node has not been enrolled to any publication lifecycle yet. Do you want to enrol it in a lifecycle?". A selector lists the available lifecycles names. If the user selects "yes", then the Publication Service is called to enrol appropriately the Node. The publication dialog is immediately displayed after that.

The dialog has two tabs. The first tab is entitled "State". The second tab is entitled "History".

In the "State" tab, the user will see the following things:

- Name of the node,
- Name of the publication lifecycle in which the Node is enrolled,
- Name of the current state. That name should be in the locale of the current user, this means that there should be internationalization resource bundles to translate the state identifier into a specific language. To do so, resource bundles keys are used like this : publication.plugin name.state name,
- Image illustrating the current state,
- Specific WebUI Form to manipulate the current state.

In the "History" tab, the user will see a table listing all history log entries.

9.6. ECM Publication Action

1. new generic publication action will be created. This one will be called "ChangePublicationState". It will

change the publication state of a content Node and will allow the following parameters:

- lifecycleName : specifies the name of publication lifecycle,
- newState : specifies the name of the new state,
- context : specifies optional information to be used when invoking the changeState() method in the publication plugin.

Basically, the "lifecycleName" parameter is only used in case the newState parameter is set to "enrolled". In that particular case, the action will enrol the Node into the specified lifecycle. In all other cases, the used lifecycle is obtained from the mixin bound to the Node and the changeState() method in the publication plugin is called.

Possible use cases for this action are:

- Each document in a specific JCR location should be enrolled with a lifecycle. In that case, a publication action is set in "add mode" into that location. It will be configured with newState set to "enrol". The lifecycleName parameter will contain the lifecycle in which the Node should be enrolled.
- The user needs to put a document into a specified JCR location to publish it. In that case, a publication action is set in "add mode" into that location. It will be configured with the newState to "published".
- The user needs to publish a document by right clicking on it in JCR File Explorer. In that case, a publication action is set in "read mode" on that document. It will change the current state to "published" for example.

9.7. ECM Action Type Management

9.8. Overview

Actions are activated when event is occurred in the ECM Explorer portlet. Thanks the UI in the explorer as described in the "Actions Concept" section, actions are bound to JCR Nodes . As you can see there we provide some default preconfigured actions such as the sendmailAction or the trasformBinaryToText one.

Each action is in fact a custom NodeType that extends one of the exo:action NodeType. Each time you want to create a new ActionType you will have to extend one of the provided concrete "exo:action" definitions. By default there exist 3 NodeTypes that extend the "exo:action" one as shown in the next diagram:

Thanks to Observation in JCR,with each change in the workspace, it monitors and dispatches registed events. When registering event, we should notice some attributes:

- **isDeep** = true|false: event affect to child node or not.
- **uuid** : array of uuids of associated parent nodes can receive event.
- **nodeTypeName**: array of node type of associated parent nodes can receive event.
- **uuid** and **nodeTypeName** are null then no restriction for receiving event.

You already knew that activating an action would, underneath, launch a BusinessProcess, a script or a Business

Rule. Now you know why! PLugin a new default type that the admin can use to create new concreate ActionType is a developer work as he will need to plug new eXo plugins to tell the service that new action types are supported.

When you reach from the ECM Admin portlet, in the "Manage ActionType" section you will get a list of al the existing ActionType as well as, in the second column, the list of the NodeType it exten,ds. That provides you the same information as in the previous picture. Indeed, we can see that the two ActionTypes that are "exo:workflowAction" and "exo:backupAction" extends the "exo:businessProcessAction" one and hence they will launch some business processes.

As ActionTypes are NodeTypes, it is not possible to remove them (for integrity reasons) but you can create new ones. Remind that you will create NodeType and not Node instances! In the next screenshot, the admin must choose among the 3 ActionType to extend. In our current case, it is a "exo:businessProcess" one. The form below the selectbox is adapted according to the selected ActionType and in the case of a business process base action type then the admin is allowed to select the business process to map with the ActionType (here "content-backup"). The admin can also define new varailable that will be then passed to the business process and stored as JCR properties when the ActionType will be instantiated.

Finally, the Move checkbox tells if the Action will move the document from one location in a workspace to another one in another workspace. Underneath, the newly created ActionType also gets the "exo:move" MixinType which defines the "exo:destWorkspace" and the "exo:destPath" JCR properties.

In the next screenshot you can see that the form is adapted to the main selected ActionType. Here we see all the available business processes that the action could launch. If the admin had selected the "exo:scriptAction" base NodeType then the list would have been filled with the available Scripts.

The next code sample is simple the "TransformBinaryChildrenToTextScript.groovy". As you can see, it gets the information through the context object and then has access to all the necessary information to process some document manipulations such as transforming all the binary documents of a folder into text documents under the same folder using the eXo DocumentReaderService.

```
import java.util.Map;

import javax.jcr.Property;
import javax.jcr.Node ;
import javax.jcr.NodeIterator;
import javax.jcr.PathNotFoundException;

import org.exoplatform.services.cms.scripts.CmsScript;
import org.exoplatform.services.document.DocumentReaderService;

/*
* Will need to get The MailService when it has been moved to exo-platform
*/
public class TransformBinaryChildrenToTextScript implements CmsScript {

  private DocumentReaderService readerService_;

  public TransformBinaryChildrenToTextScript(DocumentReaderService readerService) {
    readerService_ = readerService;
  }

  public void execute(Object context) {
    Map variables = (Map) context;

    Node actionNode = (Node) variables.get("actionNode");
    Node folderNode = actionNode.getParent();

    try {
      NodeIterator iter = folderNode.getNodes();
      while(iter.hasNext()) {
        Node childNode = iter.nextNode();
        if("nt:file".equals(childNode.getPrimaryNodeType().getName())) {
          Node content = childNode.getNode("jcr:content");
```

```
          Property mime = content.getProperty("jcr:mimeType");
          if (!mime.getString().startsWith("text")) {
            String text = readerService_.getContentAsText(mime.getString(), content
              .getProperty("jcr:data").getStream());
            Node file = null;
            try {
              file = folderNode.getNode(childNode.getName() + ".txt");
            } catch (PathNotFoundException e) {
              file = folderNode.addNode(childNode.getName() + ".txt", "nt:file");
            }
            Node contentNode = file.addNode("jcr:content", "nt:resource");
            contentNode.setProperty("jcr:encoding", "UTF-8");
            contentNode.setProperty("jcr:mimeType", "text/html");
            contentNode.setProperty("jcr:data", text);
            contentNode.setProperty("jcr:lastModified", new GregorianCalendar());
          }
        }
      }
      folderNode.save();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void setParams(String[] params) {}

}
```

Note that when a node is bound with an action then it gets the MixinType "exo:actionable" which mandates the Node to have a child Node of type exo:action. In the JCR explorer, those are, by default, managed as hidden files but you can view them by configuration from the Preferences form.

9.9. Event Type

Available Event types now are add node, remove node, add property, remove property, update property. From DMS 2.6, Each action can have more than one event type. For example: sendmailAction can be registered with events: add node, remove node, ... Thanks to ObservationManager from JCR, listener of action will be fired if its events matches event occurring. DMS have two more event types is read and schedule.

9.10. Affected node types

With DMS 2.6, when listener of action is executed, the first of all, it will check whether node making event has node type in set of affected node types to continue launch or stop. When registering one action, affected node types contains set values of node types and is kept as property of action node. If this property does not exist or is empty then above checking will be ignored. This property **exo:affectedNodeTypeNames* is in** mix:affectedNodeTypes*:

```
  <nodeType name="mix:affectedNodeTypes" isMixin="true" hasOrderableChildNodes="false" primaryItemName="">
    <propertyDefinitions>
      <propertyDefinition name="exo:affectedNodeTypeNames" requiredType="String" autoCreated="false" mandatory="
        onParentVersion="COPY" protected="false" multiple="true">
        <valueConstraints/>
      </propertyDefinition>
    </propertyDefinitions>
  </nodeType>
```

9.11. Service Configuration

```
<component>
  <key>org.exoplatform.services.cms.actions.ActionServiceContainer</key>
  <type>org.exoplatform.services.cms.actions.ActionServiceContainerImpl</type>
  ..............
</component>
```

As in many cases the service can get plugins and there are alot of defined actions type pluged-in action service thought actions plugin

- Sample action configuration:

```
     <component-plugin>
        <name>exo:scriptAction</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.cms.actions.ScriptActionPlugin</type>
        <init-params>
          <object-param>
            <name>predefined.actions</name>
            <description>description</description>
            <object type="org.exoplatform.services.cms.actions.ActionConfig">
              <field  name="workspace"><string>production</string></field>
              <field  name="actions">
                <collection type="java.util.ArrayList">
                  <value>
                    <object type="org.exoplatform.services.cms.actions.ActionConfig$Action">
                      <field  name="type"><string>${action type}</string></field>
                      <field  name="name"><string>${action name}</string></field>
                      <field  name="description"><string>${description}</string></field>
                      <field  name="srcWorkspace"><string>${workspace}</string></field>
                      <field  name="srcPath"><string>${path}</string></field>
                      <field  name="isDeep"><string>${true|false}</string></field>
                      <field  name="uuid"><string>${array uuid}</string></field>
                      <field  name="nodeTypeName">
                        <collection type="java.util.ArrayList">
                          <value><string>${nodeTypeName}</string></value>
                        </collection>
                      </field>
                      <field  name="lifecyclePhase">
                        <collection type="java.util.ArrayList">
                          <value><string>${lifecycle (node_added,node_remove...}</string></value>
                        </collection>
                      </field>
                      <field  name="roles"><string>${list of permission can execute this action}</string></fie
                      <field  name="mixins">
                        <collection type="java.util.ArrayList">
                         <value>
                              <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                           <field  name="name"><string>mix:affectedNodeTypes</string></field>
                           <field  name="properties">
                            <string>exo:affectedNodeTypeNames=exo:article,
                                 exo:podcast,exo:sample,kfx:document,nt:file,rma:filePlan</string>
                           </field>
                          </object>
                         </value>
                        </collection>
                      </field>
                      <field  name="variables">
                        <!--
                               Map necessary variables to execute action
                        -->
                      </field>
                    </object>
                  </value>
                </collection>
              </field>
            </object>
          </object-param>
        </init-params>
      </component-plugin>
```

- Script action plugin: init a send mail script action. In DMS, by default, **isDeep** = true, **uuid** = null, **nodeTypeName** = null.

```
      <component-plugin>
         <name>exo:scriptAction</name>
         <set-method>addPlugin</set-method>
         <type>org.exoplatform.services.cms.actions.ScriptActionPlugin</type>
         <init-params>
           <object-param>
             <name>predefined.actions</name>
             <description>description</description>
             <object type="org.exoplatform.services.cms.actions.ActionConfig">
               <field  name="workspace"><string>production</string></field>
               <field  name="actions">
                 <collection type="java.util.ArrayList">
                   <value>
                     <object type="org.exoplatform.services.cms.actions.ActionConfig$Action">
                       <field  name="type"><string>exo:sendMailAction</string></field>
                       <field  name="name"><string>sendMail</string></field>
                       <field  name="description"><string>send a notification mail</string></field>
                       <field  name="srcWorkspace"><string>production</string></field>
                       <field  name="srcPath"><string>/cms/publications</string></field>
                       <field  name="isDeep"><string>true</string></field>
                       <field  name="lifecyclePhase">
                         <collection type="java.util.ArrayList">
                           <value><string>read</string></value>
                         </collection>
                       </field>
                       <field  name="roles"><string>*:/admin</string></field>
                       <field  name="variables">
                         <string>exo:to=benjamin.mestrallet@exoplatform.com</string>
                       </field>
                       <field  name="mixins">
                         <collection type="java.util.ArrayList">
                          <value>
                               <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                            <field  name="name"><string>mix:affectedNodeTypes</string></field>
                            <field  name="properties">
                             <string>exo:affectedNodeTypeNames=exo:article,
                                   exo:podcast,exo:sample,kfx:document,nt:file,rma:filePlan</string>
                            </field>
                           </object>
                          </value>
                         </collection>
                       </field>
                     </object>
                   </value>
                 </collection>
               </field>
             </object>
           </object-param>
         </init-params>
       </component-plugin>
```

9.12. ECM Groovy Scripts Management

9.13. Overview

As you have probably already noticed, there are many different types of ECM groovy scripts. They all implement the CmsScript interface, which you can see here:

```
package org.exoplatform.services.cms.scripts;

public interface CmsScript {

  public void execute(Object context);
```

```
  public void setParams(String[] params);
}
```

Groovy scripts have been developed for different purposes. Therefore we have split the organization in several areas. You can find the scripts in the ECM Admin section dedicated to script management (*.../webapps/portal/WEB-INF/conf/ecm/artifacts/scripts/*).

There are two script areas:

- ECM File Explorer portlet scripts and
- Browse Content portlet scripts.

9.14. ECM File Explorer Scripts

There are three different script types used in the ECM Explorer:

- **Action scripts** are launched when an ECM action triggers them (refer to Actions Concept section for more information).
- **Interceptor scripts** are be triggered before and/or after a JCR node is saved, when a node is created or edited. They are used to either **validate** the value entered in a form or to **manipulate** the newly created node. For example to map the new node with a forum thread or any other type of discussion areas.
- **Widget scripts** are used to fill widgets - such as a select box - in a dynamic way.

The ECM Admin User Interface allows the administrator to manage all different script types through a single interface. You can edit scripts or create new ones. When creating a new script you choose one of the above-named script types that later cannot be changed.

9.15. Browse Content Portlet Scripts

The Browse Content Portlet is used to provide the end user with several ways to extract the content from the JCR. One of these ways uses a Groovy script that is responsible for the extraction.

Note that scripts are stored directly in the JCR tree in the production workspace at the path /jcr:system/exo:ecm/scripts. Scripts are written in Groovy and the JCR path there can be seen as the classpath as we modified the default Groovy classloader to look for the scripts directly in the relative JCR path. Whenever a script is modified the groovy classloader is reset.

9.16. Manage Script Service

Script service that allows to manage groovy scripts. As you can see in the configuration file you define the scripts that are loaded into the JCR. Be aware that the composition of the folder name of each script defines the script type!

You find the configuration at *.../portal/WEB-INF/conf/ecm/ecm-scripts-configuration.xml*.

```
    <component>
      <key>org.exoplatform.services.cms.scripts.ScriptService</key>
      <type>org.exoplatform.services.cms.scripts.ScriptServiceImpl</type>
      ..................
    </component>
...
      <init-params>
        <object-param>
          <name>predefined.scripts</name>
          <description>description</description>
          <object type="org.exoplatform.services.cms.impl.ResourceConfig">
            <field  name="workspace"><string>production</string></field>
            <field  name="resources">
              <collection type="java.util.ArrayList">
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/widget/FillSelectBoxWithWorkspaces.groovy</string><
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/action/SendMailScript.groovy</string></field>
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/action/RSSScript.groovy</string></field>
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/widget/SetJCRBrowserForProductionWorkspaceAndRulesP
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/widget/SetJCRBrowserForProductionWorkspaceAndScript
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/widget/FillSelectBoxWithNodeChildren.groovy</string
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/action/TransformBinaryChildrenToTextScript.groovy</
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/interceptor/PreNodeSaveInterceptor.groovy</string><
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>ecm-explorer/interceptor/PostNodeSaveInterceptor.groovy</string>
                  </object>
                </value>
                <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                    <field  name="name"><string>content-browser/GetDocumentsFromWorkspaceAndPath.groovy</string>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </object-param>
      </init-params>
```

9.17. ECM Template Management

9.18. Overview

The templates are applied to a NodeType or a metadata MixinType. Two kinds of templates exist :

- **dialogs** : are html forms that allow to create node instances
- **views** : are html fragments used to display nodes

From the ECM admin portlet, *Manage Template* lists existing NodeTypes that have been associated to Dialog and/or View templates. These templates can be attached to permissions (in the usual *membership:group_ form), so that a specific one is displayed according to the rights of the user (very useful in a content validation workflow activity).*

9.19. DocumentType

1. checkbox allows to say if the nodetype should be considered as a **DocumentType**. File Explorer considers such nodes as user content and applies the following behavior :

- View template will be used to display the DocumentType nodes
- DocumentTypes nodes can created by the '*Add Document*' action
- non DocumentType are hidden (unless *'Show non document types'* option is checked)

Templates are written using Groovy Templates and will require some experience with JCR API and HTML notions.

9.20. The Dialog Syntax

Dialogs are groovy templates that generate forms by mixing static HTML fragments and groovy calls to the components responsible for building the UI at runtime. The result is a simple but powerfull syntax.

9.20.1. Interceptors

By placing interceptors in your template, you will be able to execute a groovy script just before and just after node save. Pre node-save interceptors are mostly used to validate input values and their overall meaning while the post node-save interceptor can be used to do some manipulation or reference for the newly created node such as binding it with a forum discussion or wiki space.

To place interceptors, use the following fragment :

```
<% uicomponent.addInterceptor("ecm-explorer/interceptor/PreNodeSaveInterceptor.groovy", "prev");%>
```

Interceptor groovy scripts are managed in the '*Manage Script*' section in the ECM admin portlet. They must implement the **CmsScript** interface. Pre node-save are passed input values within the context :

```
public class PreNodeSaveInterceptor implements CmsScript {

  public PreNodeSaveInterceptor() {
  }

  public void execute(Object context) {
    Map inputValues = (Map) context;
    Set keys = inputValues.keySet();
    for(String key : keys) {
      JcrInputProperty prop = (JcrInputProperty) inputValues.get(key);
      println("   --> "+prop.getJcrPath());
    }
  }

  public void setParams(String[] params) {}

}
```

Whereas post node-save one are passed the path of the saved node in the context:

```
<% uicomponent.addInterceptor("ecm-explorer/interceptor/PostNodeSaveInterceptor.groovy", "post");%>
```

```
public class PostNodeSaveInterceptor implements CmsScript {

  public PostNodeSaveInterceptor() {
  }

  public void execute(Object context) {
    String path = (String) context;

    println("Post node save interceptor, created node: "+path);
  }

  public void setParams(String[] params) {}
```

9.20.2. Hidden fields

In the next code sample, each argument is composed of a set of keys and values. The order of the argument is not important and only the key matters. That example builds a field which has the id "hiddenInput4", which will generate a date (as the widget is a calendar) but which is not visible. In other words the value of the field will be automatically set to the current date value and no visible field will be printed on the form.

```
String[] hiddenField4 = ["jcrPath=/node/jcrcontent/jcr:lastModified", "visible=false"];
uicomponent.addCalendarField("hiddenInput4", hiddenField4);
```

Once the form is saved, that date value will be saved in under the relative JCR path ./exo:image/jcr:lastModified

#warning("As of ECM 2.2 and DMS 2.3 the calendar field is by default invisible! In order to see a calendar field your must add a visible=true option.") 1.1 Non value field, nodetype or mixintype creation

In many cases, when creating a Node instance out of a form, one must still tell the CMS service about the structure of that node. In other words, the template creator must tell what nodetype is a child of the newly created node or if that current node has any mixin type attributed.

By defining these arguments the node and its children are created with the correct nodetype and mixintype.

See the following exemple :

```
String[] hiddenField = ["jcrPath=/node/jcrcontent", "nodetype=nt:resource", "mixintype=exo:rss-enable", "visible
 uicomponent.addHiddenField("hiddenInput", hiddenField) ;
```

1.1 Hidden field with default value

In the first sample the widget was a calendar one and the value was automatically created and set according to the current date. In that new example we show that it is also possible to set a default value for a field.

Furthermore, as no widget is specified then a text widget is used. Here too the widget is not visible.

```
String[] hiddenField = ["jcrPath=/node/jcrcontent/jcr:mimeType", "image/jpeg"] ;
uicomponent.addHiddenField("hiddenInput", hiddenField) ;
```

1.1 Non editable and visible if not null fields

It is possible to create widgets that are non editable (and then only used to print some information). Furthermore, it is possible to tell that a widget should be visible only if its value is not null, in other words when the form is used to edit an already existing node.

```
String nameArgs[] = ["jcrPath=/node", "mixintype=mix:votable", "visible=if-not-null"];
uicomponent.addMixinField("name", nameArgs )
```

1.1 WYSIWYG widget

The "What You See Is What You Get" widget is one of the most powerful one. It prints an advanced javascript text editor with many functionalities including the ability to dynamically upload images or flash assets into a JCR workspace and then to reference them from the created HTML text.

```
String[] fieldSummary = ["jcrPath=/node/exo:summary", "options=basic"] ;
uicomponent.addWYSIWYGField("summary", fieldSummary) ;
```

The "options" argument is used to tell to the component which toolbar should be used (a toolbar is a set of buttons that the editor will render).

By default there are two options:

- **basic**: a minimal set of icons is printed
- **default**: a large set of icons is printed, no options argument is needed in that case

Note that there is also a simple textarea widget:

```
String [] descriptionArgs = ["jcrPath=/node/exo:title", "validate=empty"];
uicomponent.addTextAreaField("description", descriptionArgs) ;
```

1.1 A calendar widget with time

We already introduced the calendar widget, that widget can be extended with the use of the options = displaytime argument to print some time.

```
String [] dateArgs = ["jcrPath=/node/exo:date", "options=displaytime"];
uicomponent.addCalendarField("date", dateArgs) ;
```

#warning("As of ECM 2.2 and DMS 2.3 the calendar field is by default invisible! In order to see a calendar field your must add a visible=true option.") 1.1 Upload widget

The dialog syntax also provides an upload widget which allows the user to upload a document with the generated form. Once again the syntax is very simple and the result is impressive.

```
String [] imageArgs = ["jcrPath=/node/exoimage/jcr:data"];
uicomponent.addUploadField("image", imageArgs);
```

1.1 Simple selectbox widget

We also provide a selectbox widget with a simple default mechanism where we can print all the available static options, separated with comma, in the "options" argument of the directive. The argument with no key (here "text/html") is the selected option of the list.

```
String[] mimetype = ["jcrPath=/node/jcrcontent/jcr:mimeType", "text/html", "options=text/html,text/plain"] ;
uicomponent.addSelectBoxField("mimetype", mimetype) ;
```

As usual, the value will be stored at the relative path defined by the jcrPath directive argument.

1.1 Advanced dynamic select box

In many cases, the previous solution with static options is not good enough and one would like to have the select list filled dynamically. That is what we provide thanks to the introduction of a Groovy script as shown in the next code fragment.

```
String[] args = ["jcrPath=/node/exodestWorkspace", "script=ecm-explorer/widget/FillSelectBoxWithWorkspaces:groov
uicomponent.addSelectBoxField("destWorkspace", args) ;
```

The script itself implements the CmsScript interface and the cast is done to get the select box object as shown in the script code which fill the select list with the existing JCR workspaces.

```
import java.util.List ;
import java.util.ArrayList ;

import org.exoplatform.services.jcr.RepositoryService;
import org.exoplatform.services.jcr.core.ManageableRepository;

import org.exoplatform.webui.form.UIFormSelectBox;
import org.exoplatform.webui.core.model.SelectItemOption;
import org.exoplatform.services.cms.scripts.CmsScript;

public class FillSelectBoxWithWorkspaces implements CmsScript {

  private RepositoryService repositoryService_;

  public FillSelectBoxWithWorkspaces(RepositoryService repositoryService) {
    repositoryService_ = repositoryService;
  }

  public void execute(Object context) {
    UIFormSelectBox selectBox = (UIFormSelectBox) context;

    ManageableRepository jcrRepository = repositoryService_.getRepository();
    List options = new ArrayList();
```

```
    String[] workspaceNames = jcrRepository.getWorkspaceNames();
    for(name in workspaceNames) {
      options.add(new SelectItem(name, name));
    }
    selectBox.setOptions(options);
  }

  public void setParams(String[] params) {}
```

Finally note that it is also possible to provide a parameter to the script by using the argument "scriptParams"

1.1 Widget with selector

One of the most advanced functionality of this syntax is the ability to plug your own component that shows an interface to help the user choose the value of the field.

In the generated form you will see an icon. This icon is configurable thanks to the **selectorIcon** argument.

The syntax is a bit more complex but not much.

```
String[] groupArgs = ["jcrPath=/node/exogroup",
    "selectorClass=org:exoplatform:ecm:webui:selector:UIGroupMemberSelector"];
uicomponent.addActionField("group", groupArgs);
```

You can plug your own component using the selectorClass argument. It just must follow eXo UIComponent mechanism and implements the interface ComponentSelector:

```
package org.exoplatform.ecm.webui.selector;

import org.exoplatform.webui.core.UIComponent;

/**
 * Created by The eXo Platform SARL Author : Dang Van Minh
 * minh.dang@exoplatform.com Nov 16, 2006
 */
public interface ComponentSelector {

  /**
   * Gets the source component of a selector
   *
   * @return the source component
   */
  public UIComponent getSourceComponent();

  /**
   * Sets the source component of a selector
   *
   * @param uicomponent the uicomponent
   * @param initParams the init params
   */
  public void setSourceComponent(UIComponent uicomponent, String[] initParams) ;
}
```

1.1 Selector that takes parameters

A Selector component can get parameters (they are passed thanks to the initParams] argument of the method setSourceComponent). The argument used to give a parameter is the selectorParams one.

1.1 Multivalue widget

A widget can be a multiple value one if you add the argument "multiValues=true" to the directive.

1 Manage template service

Template service allows to create dialogs and view templates for each node type registered. Each node type may have many dialogs and view templates. The template will be used when creating or viewing nodes.

.../webapps/portal/WEB-INF/conf/ecm/ecm-templates-configuration.xml

```
<component>
    <key>org.exoplatform.services.cms.templates.TemplateService</key>
    <type>org.exoplatform.services.cms.templates.impl.TemplateServiceImpl</type>
    ..............
  </component>
```

As usual one can register a plugin inside the service. This plugin init default dialogs and views template of any node type as nt:file, exo:article, exo:workflowAction, exo:sendMailAction, ...

```
<component-plugins>
        <component-plugin>
          <name>addTemplates</name>
          <set-method>addTemplates</set-method>
          <type>org.exoplatform.services.cms.templates.impl.TemplatePlugin</type>
          .........
        </component-plugin>
      </component-plugins>
```

With init parameters as:

```
<init-params>
        <value-param>
                <name>autoCreateInNewRepository</name>
                <value>true</value>
        </value-param>
        <value-param>
                <name>storedLocation</name>
                <value>war:/conf/ecm/artifacts/templates</value>
        </value-param>
        <value-param>
                <name>repository</name>
                <value>repository</value>
        </value-param>
  <object-param>
        <name>template.configuration</name>
        <description>configuration for the localtion of templates to inject in jcr</description>
        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig">
          <field  name="nodeTypes">
                <collection type="java.util.ArrayList">
                  <value>
                        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType">
                          <field  name="nodetypeName"><string>exo:article</string></field>
                          <field  name="documentTemplate"><boolean>true</boolean></field>
                          <field  name="label"><string>Article</string></field>
                          <field  name="referencedView">
                                <collection type="java.util.ArrayList">
                                  <value>
                                        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig
                                          <field name="templateFile"><string>/article/views/view1.gtmpl</string>
                                          <field name="roles"><string>*</string></field>
                                        </object>
                                  </value>
                                </collection>
                          </field>
                          <field  name="referencedDialog">
                                <collection type="java.util.ArrayList">
                                  <value>
                                        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig
                                          <field name="templateFile"><string>/article/dialogs/dialog1.gtmpl</str
                                          <field name="roles"><string>*</string></field>
                                        </object>
                                  </value>
                                </collection>
```

```
                        </field>
                      </object>
                  </value>
                </collection>
            </field>
          </object>
  </object-param>
</init-params>
```

9.21. UI Extension Framework

9.22. Overview

Since DMS 2.5, it is possible to extend the **File Explorer** and the **ECM Administration** thanks to the **UI Extension Framework**. Indeed, you can add your own action buttons to the **File Explorer** and/or add your own managers to the **ECM Administration**.

In this tutorial, we will see how to add a new action button to the **File Explorer** but the logical is exactly the same for the managers in the **ECM Administration**.

Note

For this tutorial, you will need java 5, maven 2 and eclipse

9.23. How to add a new action button to the File Explorer?

9.23.1. Create your UI Action

9.23.1.1. Create the UI Action class

Create a **pom.xml** file from the following content

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.exoplatform.ecm.dms.core</groupId>
  <artifactId>exo.ecm.dms.core.portlet.ecm.tutorial.main</artifactId>
  <packaging>jar</packaging>
  <name>UI Extension Framework Tutorial Classes</name>
  <version>trunk</version>
  <url>http://www.exoplatform.org</url>
  <description>UI Extension Framework Tutorial Classes</description>

  <dependencies>
    <dependency>
      <groupId>org.exoplatform.ecm.dms.core</groupId>
      <artifactId>exo.ecm.dms.core.webui.ext</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.exoplatform.ecm.dms.core</groupId>
      <artifactId>exo.ecm.dms.core.portlet.ecm.core.main</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
```

```
  <build>
    <resources>
      <resource>
        <directory>src/main/java</directory>
      </resource>
    </resources>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
                <source>1.5</source>
                <target>1.5</target>
                <optimize>true</optimize>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Create the directories src/main/java and launch **mvn eclipse:eclipse* You can then, launch your eclipse and import this new project**

Create a new class called org.exoplatform.ecm.tutorial.MyUIAction that extends org.exoplatform.webui.core.UIComponent

1.1.1 Create your ActionListener

The webui framework allows you to be notified when a given action has been triggered, you just need to call your action listener as follow $ACTION*NAME}ActionListener*

So, if you want to call our action MyActionName, the listener will be then called MyActionNameActionListener.

So first, add a static inner class called MyActionNameActionListener that extends org.exoplatform.ecm.webui.component.explorer.control.listener.UIActionBarActionListener<MyUIAction>

See below the expected code

```
public static class MyActionNameActionListener extends UIActionBarActionListener<MyUIAction> {
        @Override
        protected void processEvent(Event<MyUIAction> event) throws Exception {
                System.out.println("MyActionName of the component MyUIAction has been triggered!!");
        }
}
```

Note

The UIActionBarActionListener is the super class of all the ActionListeners related to the action bar, this class sets internaly the context of an action button

Then, define the action listener at the class definition level with the webui annotations as below:

```
@ComponentConfig(
     events = {
       @EventConfig(listeners = MyUIAction.MyActionNameActionListener.class)
     }
 )
public class MyUIAction extends UIComponent {
```

1.1.1 Deploy your Action

Launch **mvn clean install** and copy the file target/exo.ecm.dms.core.portlet.ecm.tutorial.main-trunk.jar into $TOMCAT*HOME}/webapps/ecm/WEB-INF/lib*

1.1 Register your UI Action

To register your action, you need to create a dedicated project that will manage the configuration

1.1.1 Create the configuration file

Create a **pom.xml** file from the following content

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.exoplatform.ecm.dms.core</groupId>
  <artifactId>exo.ecm.dms.core.portlet.ecm.tutorial.config</artifactId>
  <packaging>jar</packaging>
  <name>UI Extension Framework Tutorial Config</name>
  <version>trunk</version>
  <url>http://www.exoplatform.org</url>
  <description>UI Extension Framework Tutorial Config</description>

  <build>
    <resources>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.xml</include>
        </includes>
      </resource>
    </resources>
  </build>
</project>
```

Create the directories *src/main/java* and launch **mvn eclipse:eclipse* You can then, launch your eclipse and import this new project**

Create a new configuration file *conf/portal/configuration.xml* to register your action with the service *org.exoplatform.webui.ext.UIExtensionManager*, as below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
  <external-component-plugins>
    <!- The full qualified name of the UIExtensionManager that manages all the UI Extensions ->
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>add.action</name>
      <!- The method to call to register a new UI Extension ->
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>MyActionName</name>
          <object type="org.exoplatform.webui.ext.UIExtension">
            <!- The type of extension, here it is the type related to the Action Bar in the File Explorer ->
            <field name="type"><string>org.exoplatform.ecm.dms.UIActionBar</string></field>
            <!- The name of your action ->
            <field name="name"><string>MyActionName</string></field>
            <!- The full qualified name of your webui component ->
            <field name="component"><string>org.exoplatform.ecm.tutorial.MyUIAction</string></field>
          </object>
        </object-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

1.1.1 Deploy the configuration file

Launch **mvn clean install** and copy the file target/exo.ecm.dms.core.portlet.ecm.tutorial.config-trunk.jar into $TOMCAT*HOME}/lib*

1.1 Define a label and an icon for your action

1.1.1 Define a label in ECM Administration

Edit the file $TOMCAT*HOME}/webapps/ecm/WEB-INF/classes/locale/portlet/ecm/ECMAdminPortlet*en.xml (for english which is also the default language) and add the label as below:

```
...
  <UIViewFormTabPane>
    ...
    <label>
      <myActionName>My Action Name</myActionName>
    ...
    </label>
    ...
  </UIViewFormTabPane>
...
```

9.23.1.2. Define a label in the File Explorer

Edit the file *$TOMCAT*HOME}/webapps/ecm/WEB-INF/classes/locale/portlet/ecm/JCRExplorerPortlet*en.xml* (for english which is also the default language) and add the label as below:

```
...
  <UIActionBar>
    ...
    <tooltip>
      <MyActionName>My Action Name</MyActionName>
    ...
    </tooltip>
    ...
  </UIActionBar>
...
```

1.1.1 Define a logo

Edit the file $TOMCAT*HOME}/webapps/ecm/skin/icons/24x24/DefaultStylesheet.css (for the default Skin) and add the icon definition as below (in this case, we re-use the "Add Document" icon but you could add your own picture into the directory $TOMCAT*HOME}/webapps/ecm/skin/icons/24x24/DefaultSkin):

```
...
.MyActionNameIcon {
  height: 24px;
  background: url('DefaultSkin/AddDocument.gif') no-repeat;
}
...
```

9.23.2. Test your Action

- Launch tomcat
- Sign in as root

- Go to the **ECM Administration**
- Choose *Manage View*
- Edit the *admin-view*
- Select the tab *actions*
- Check the you action *My Action Name* in the list

- Save the Tab change
- Save the View change
- Sign out/Sign in to force the reload of your **File Explorer**
- Go to the **File Explorer**
- Choose the drive *Collaboration Center*
- Select the tab *Actions*

You should see your action, you can then click on your button and see the message "MyActionName of the component MyUIAction has been triggered!!" in your log trace.

9.24. How to filter an UI Extension?

9.24.1. Filters Overview

In the UI Extension framework, you can use internal and external filters. The internal filters are part of the business logic of your component, for example if your component is only dedicated to articles, you will add an internal filter to your component that will check the type of the current document. The external filters are mainly used to add new filters that are not related to the business logic, to your component, a good example is the UserACLFilter which allows you to filter by access permissions.

For each filter, you must set its type. The types are defined in the interface *org.exoplatform.webui.ext.filter.UIExtensionFilterType*

```
public enum UIExtensionFilterType {

  /**
   * This type of filter will be used to know if the
   * action related to the extension can be launched
   * and to know if the component related to the
   * extension can be added to the webui tree
   * The filter is required to launch the action and
   * to add the component related to the extension
   * to the webui tree.
   * If it succeeds, we will check the other filters.
   * If it fails, we will stop.
   */
  MANDATORY,
  /**
   * This type of filter will be used to know if the
   * action related to the extension can be launched.
   * The filter is required to launch the action.
```

```
   * to the webui tree.
   * If it succeeds, we will check the other filters.
   * If it fails, we will stop.
   */
  REQUISITE,
  /**
   * This type of filter will only be used to know if the
   * action related to the extension can be launched.
   * The filter is required to launch the action.
   * If it succeeds or fails, we will check the other filters.
   * It can be used to add warnings
   */
  REQUIRED,
  /**
   * This type of filter will only be used to know if the
   * action related to the extension can be launched.
   * The filter is not required to launch the action.
   * If it succeeds or fails, we will check the other filters.
   * It can be used for auditing purpose
   */
  OPTIONAL
}
```

1.1 Add an internal Filter

Let's add a MANDATORY filter to our component that will check if the document is of type exo:article~.

First create a new class called org.exoplatform.ecm.tutorial.MyUIFilter that implements org.exoplatform.webui.ext.filter.UIExtensionFilter

See below the code of our example

```
public class MyUIFilter implements UIExtensionFilter {

        /**
         * This method checks if the current node is of the right type
         */
        public boolean accept(Map<String, Object> context) throws Exception {
                // Retrieve the current node from the context
            Node currentNode = (Node) context.get(Node.class.getName());
                return currentNode.isNodeType("exo:article");
        }

        /**
         * This is the type of the filter
         */
        public UIExtensionFilterType getType() {
                return UIExtensionFilterType.MANDATORY;
        }

        /**
         * This is called when the filter has failed
         */
        public void onDeny(Map<String, Object> context) throws Exception {
                System.out.println("This document has been rejected");
        }
}
```

Note

In this example, the filter directly implements UIExtensionFilter but it can be very helpful to extend the class org.exoplatform.webui.ext.filter.UIExtensionAbstractFilter because it provides you the methods createUIPopupMessages that adds messages to the UIPopup Window

Note

The context that you get in the accept method, has been set by the class UIActionBarActionListener, in this context you can find the UIActionBar, the UIJCRExplorer, the UIApplication, the WebuiRequestContext and the Node of the current context

Once done, you just need to assign this filter to our component by using the annotation *org.exoplatform.webui.ext.filter.UIExtensionFilters* as below

```
public class MyUIAction extends UIComponent {

    private static final List<UIExtensionFilter> FILTERS = Arrays.asList(new UIExtensionFilter[] {new MyUIFilter

        @UIExtensionFilters
        public List<UIExtensionFilter> getFilters() {
                return FILTERS;
        }
...
```

Launch **mvn clean install** and copy the file target/exo.ecm.dms.core.portlet.ecm.tutorial.main-trunk.jar into $TOMCAT*HOME}/webapps/ecm/WEB-INF/lib*

You can now check that your action only appears when you select an article.

1.1 Add an external Filter

If you want for example, that you action is only available for the managers of the group platform/administrators, you just need to change the configuration file as below:

```
...
<!-- The external filters -->
<field name="extendedFilters">
        <collection type="java.util.ArrayList">
                <value>
                        <object type="org.exoplatform.webui.ext.filter.impl.UserACLFilter">
                                <field name="permissions">
                                        <collection type="java.util.ArrayList">
                                                <value>
                                                        <string>manager:/platform/administrators</string>
                                                </value>
                                        </collection>
                                </field>
                        </object>
                </value>
        </collection>
</field>
...
```

Launch **mvn clean install** and copy the file *target/exo.ecm.dms.core.portlet.ecm.tutorial.config-trunk.jar* into *$TOMCAT*HOME}/lib

You can now check that your action only appears when the current user is a manager of the group *platform/administrators*.

Note

The entire projects have been attached to this wiki article

Chapter 10. Workflow

Chapter 11. Configuration

11.1. Workflow Publication Plugin

11.2. Overview

Workflow Publication Plugin allows users to publish a content of document. It breaks a work process down into tasks. A basic example of such a process is an approval workflow process, in which an employee needs a manager's permission before running an application. A workflow engine provides an infrastructure to model this workflow, execute it, assign the tasks to its participants and monitor it. To achieve the desired results, it may interact with humans or machines through, for example, Web services. This enables integration with platforms different from Java, like mainframes or .NET.

In **Workflow Publication Plugin**, each phase has its lifecycle.

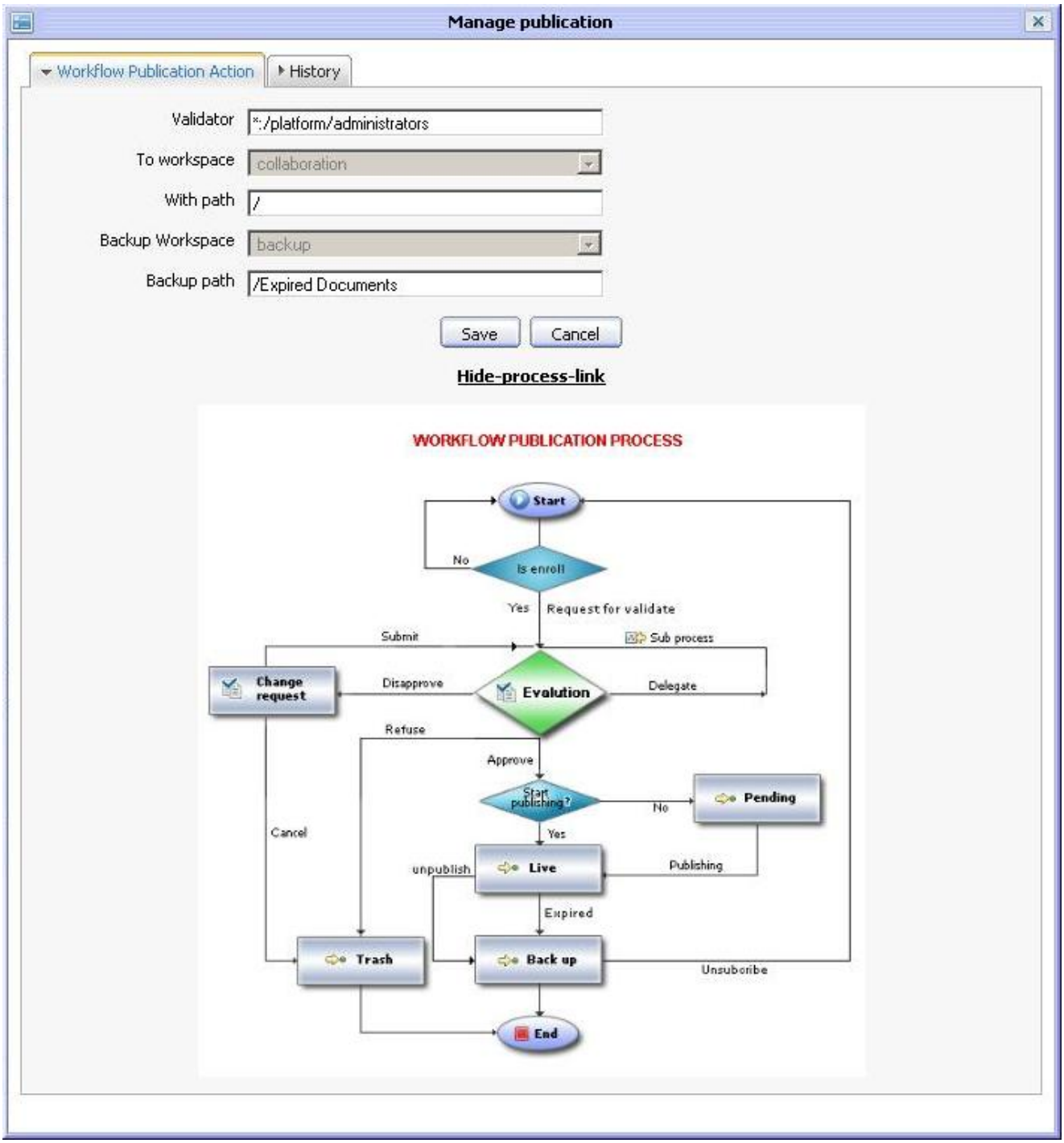
- When creating a new document, and manage publication, enrolled it, the lifecycle is **enrolled**.
- After creating workflow publication for it (choose validator, dest workspace, dest path, backup path or using default), the lifecycle is **content publishing**.
- When the validator person manage it, after approve it, the lifecycle is **published**.
- When the document is published, it can be unpublished. When the document is unpublished, it will be moved to **backup** workspace within backup path, and the lifecycle will be changed to **backup**.

It can be brought back to the primitive state by using the Unsubcribe function when the lifecycle is **backup**.

11.3. Pulication Configuration

11.3.1. Publication Configuration Interface

The Publication Configuration Interface is prefilled with the default values you defined in the configuration file.



When creating a workflow publication for a document, if the value isEditable is **false**, it will use the default value.

11.3.2. Configuration File

In the file **configuration.xml**, to add a workflow publication plugin, insert these code below:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.ecm.publication.PublicationService</target-component>
   <component-plugin>
      <name>Workflow</name>
      <set-method>addPublicationPlugin</set-method>
      <type>org.exoplatform.services.ecm.publication.plugins.workflow.WorkflowPublicationPlugin</type>
      <description>Workflow Publication</description>
      <init-params>
         <value-param>
      <name>validator</name>
```

```
      <value>*:/platform/administrators</value>
   </value-param>
   <value-param>
      <name>to_workspace</name>
      <value>collaboration</value>
   </value-param>
         <value-param>
      <name>destPath</name>
      <value>/Documents/Live</value>
   </value-param>
   <value-param>
      <name>destPath_currentFolder</name>
      <value>true</value>
   </value-param>
   <value-param>
      <name>isEditable</name>
      <value>false</value>
   </value-param>
         <value-param>
      <name>backupWorkspace</name>
      <value>backup</value>
        </value-param>
      </init-params>
   </component-plugin>
</external-component-plugins>
```

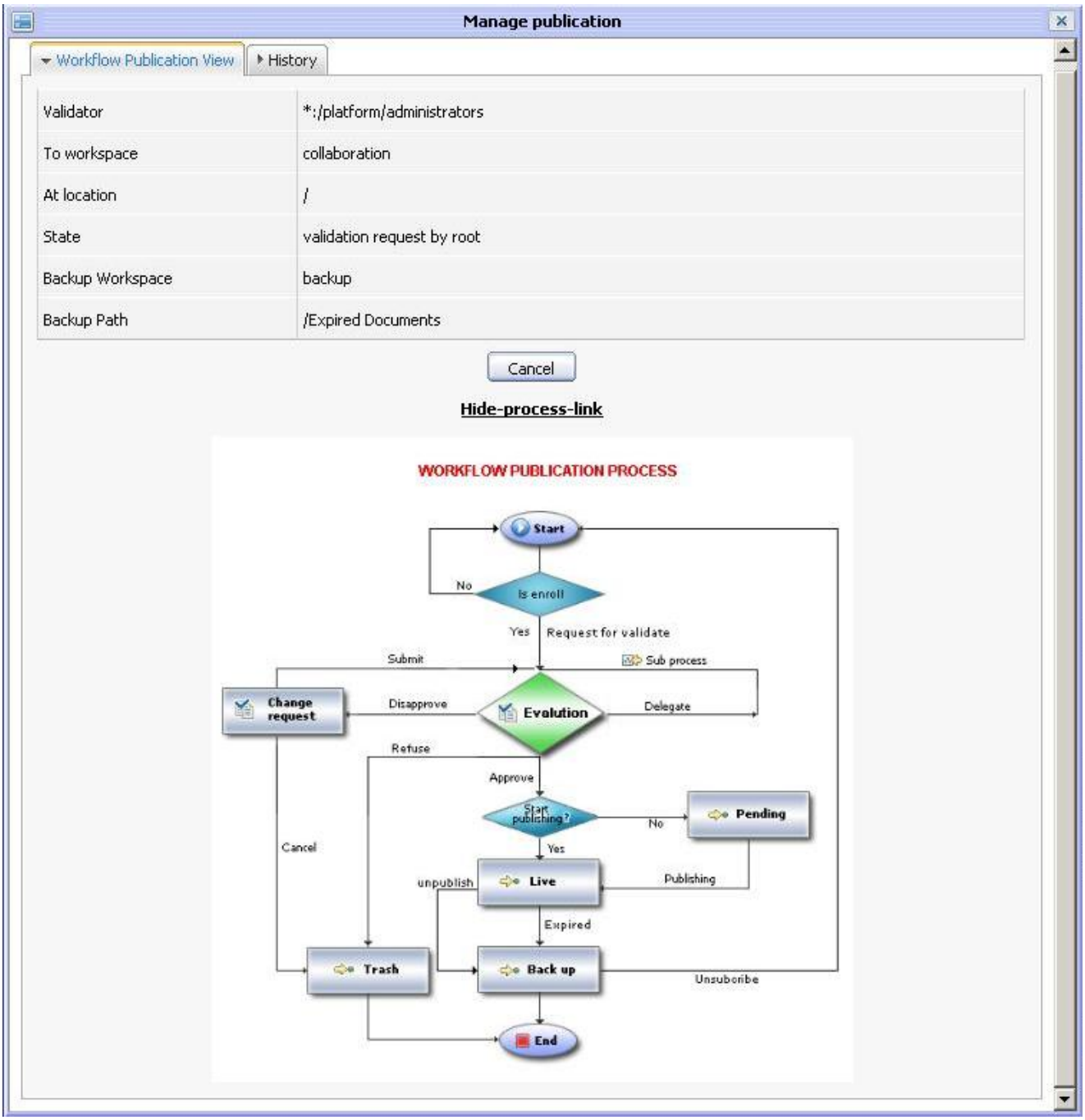
11.3.3. Configuration Parameters

Table 11.1.

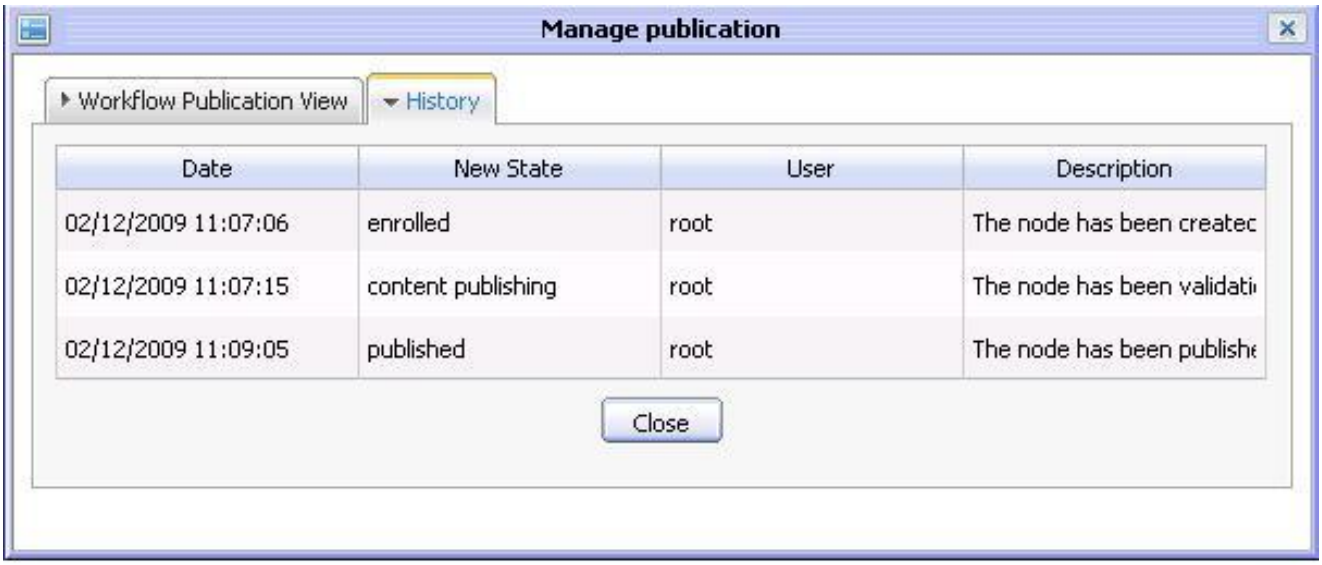
Field	Default value	Explain
validator	**:/platform/administrators**	Users have responsibility to manage tasks which are requested.
to*workspace*	collaboration	The workspace that node will be stored during the task is being evaluated.
destPath*currentFolder*	true	The parameter is specified to decide which the desPath is
destPath	The current folder if the value destPath*currentFolder is true and "/Documents/Live" if the value destPath*currentFolder is false	The path of workspace that node will be stored after being approved and then being published.
backupWorkspace	backup	The workspace that node will be removed to when user unpublish this task
backuppath	/Expired Documents	The path that node will be removed to when user unpublish this task

11.3.4. Publication Workflow View

The following image is the view after manage workflow publication for a document.



11.3.5. Example Publication History



11.3.6. Publication Nodetype Configuration

Here is the file **nodetypes-workflow-publication-config.xml**. This file is used to configure a new nodetype (nodetype workflow publication).

```
<nodeType name="publication:workflowPublication" isMixin="true" hasOrderableChildNodes="false" primaryItemName="
    <supertypes>
      <supertype>publication:publication</supertype>
    </supertypes>
    <propertyDefinitions>
      <propertyDefinition name="publication:validator" requiredType="String" autoCreated="true" mandatory="true"
        onParentVersion="COPY" protected="false" multiple="false">
        <valueConstraints/>
        <defaultValues>
          <defaultValue>*</defaultValue>
        </defaultValues>
      </propertyDefinition>
      <propertyDefinition name="publication:businessProcess" requiredType="String" autoCreated="true" mandatory=
        onParentVersion="COPY" protected="true" multiple="false">
        <valueConstraints/>
        <defaultValues>
          <defaultValue>content-publishing</defaultValue>
        </defaultValues>
      </propertyDefinition>
    </propertyDefinitions>
</nodeType>
```

The property **"publication:validator"** is used to specify the membership who can manage the workflow publication action. The property **"publication:businessProcess"** is used to specify the name of the process.

11.4. Process Definition (XPDL)

11.5. Overview

In order to take advantages of the Bonita advanced features using a XPDL based process definition, we have agreed on a couple of advanced attributes which will allow you to use entities such Hooks, mappers or performer assignments in Bonita remaining compliant with the XPDL standard. Those attributes are called in XPDL "extended attributes".

11.6. Attributes

We recommend you take a look at "holiday" processdefinition.xpdl file to see exactly where you have to add the following tags.

11.6.1. Activities attributes

Bonita allows you to create local attribute, related to a particular activity, in contrast with global attributes. The following attributes indicate that the attribute "choice" is local to the activity "eval enough holiday".

```
<DataField Id="choice" Name="choice">
  ...
  <ExtendedAttributes>
    <ExtendedAttribute Name="PropertyActivity" />
  </ExtendedAttributes>
  ...
</DataField>
...
<Activity Id="eval enough holiday" Name="eval enough holiday">
  ...
  <ExtendedAttributes>
    <ExtendedAttribute Name="property" Value="choice"/>
  </ExtendedAttributes>
  ...
</Activity>
```

11.6.2. Propagated attribute

Activities attributes can be propagated to sub-activities (daughter activities) if the "Propagated" tag is added:

```
<Activity Id="eval enough holiday" Name="eval enough holiday">
  ...
  <ExtendedAttributes>
    ...
    <ExtendedAttribute Name="property" Value="choice">
      <Propagated>Yes</Propagated>
    </ExtendedAttribute>
    ...
  </ExtendedAttributes>
  ...
</Activity>
```

11.6.3. Participants section

Some particular roles need extended attributes, please refers to section "Roles" for more details.

~~initiator: an user who started the project instance~~

```
<Participant Id="initiator" Name="initiator">
<ParticipantType Type="ROLE" />
  <ExtendedAttributes>
    <ExtendedAttribute Name="Mapper" Value="Properties" />
  </ExtendedAttributes>
</Participant>
```

~~All users in an eXo group~~

You can assign an activity to all users in a specific eXo role. eXo role starts with an eXo membership (member, owner, validator) and ends with a group: platform/administrators or organization/management/human-resources etc...

```
<Participant Id="*:/organization/management/human-resources" Name="*:/organization/management/human-resources">
          <ParticipantType Type="ROLE" />
          <ExtendedAttributes>
            <ExtendedAttribute Name="Mapper" Value="Custom" />
            <ExtendedAttribute Name="MapperClassName" Value="hero.mapper.ExoOrganizationMapper" />
            <ExtendedAttribute Name="NewParticipant" Value="true" />
            <ExtendedAttribute Name="XOffset" Value="0" />
            <ExtendedAttribute Name="YOffset" Value="0" />
          </ExtendedAttributes>
        </Participant>
```

11.6.4. Hooks

In "holiday" business process, the activity "eval enough holiday" uses a before termination hook. Here it is the corresponding extended attribute. Note that the hook class has to be in src/java/hero/hook/.

```
<Activity Id="eval enough holiday" Name="eval enough holiday">
...
  <ExtendedAttributes>
    ...
    <ExtendedAttribute Name="hook" Value="hero.hook.HolidayEnoughHolidaysLeftHook">
      <HookEventName>beforeTerminate</HookEventName>
    </ExtendedAttribute>
    ...
  </ExtendedAttributes>
...
</Activity>
```

11.6.5. Iterations

Bonita supports cycles within a process. As it's done in "holiday" business process, we define a cycle between activities "evaluation" and "change request". The iteration is defined in the last activity of the cycle. The following extended attribute is defined in "change request" activity.

```
  <ExtendedAttributes>
    <ExtendedAttribute Name="Iteration" Value="changedecision.equals("Modify")">
      <To>evaluation</To>
    </ExtendedAttribute>
  </ExtendedAttributes>
```

"Value" is the condition to continue the iteration.

11.7. Bonita Forms Definition

11.8. Overview

The concept behind Bonita forms is to generate forms corresponding to activities in the user ToDo list (for example activities runnable by the user, in ready or anticipative state). The forms present widgets that enable the user to get and change values corresponding to activity properties. Project instances properties can also be

taken into account. The forms.xml file describes forms associated with workflow activities.

11.9. Configuration

The syntax is quite simple. Between forms/forms tag which surround the file, you define forms one by one.

"Holiday" workflow has an activity called "evaluation". Let's take a look at this form definition

```
<form>
  <resource-bundle>evaluation</resource-bundle>
  <activity>evaluation</activity>
  <variable name="initiator" editable="false"/>
  <variable name="start" component="date"/>
  <variable name="end" component="date"/>
  <variable name="decision" component="radiobox"/>
  <!-- No need to specify Submit buttons in the Bonita implementation -->
</form>
```

- resource-bundle: textual information associated with the activity.

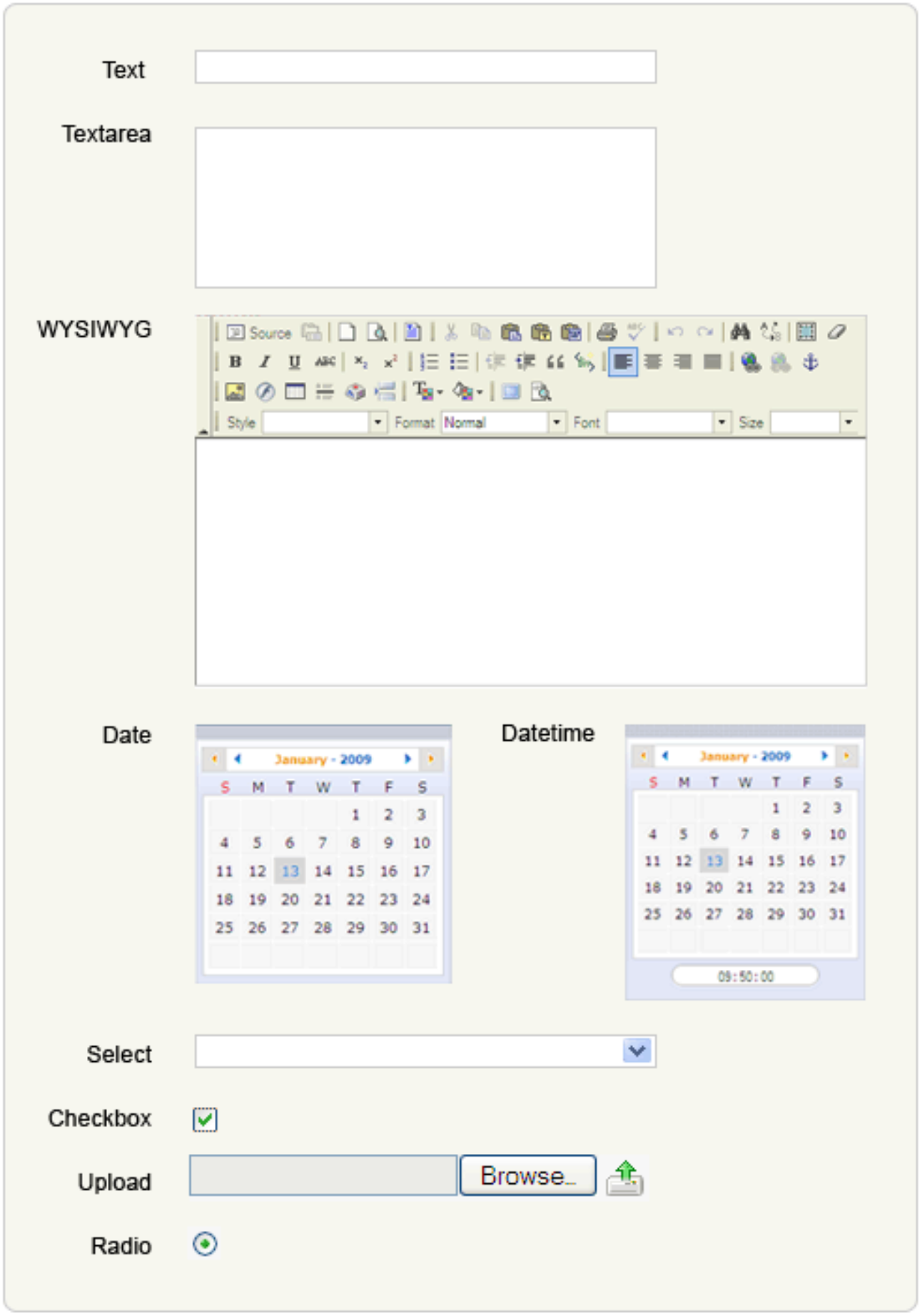
Go to "Resource bundle" section for more details.

- state-name: workflow activity name, as specified in the XPDL file
- variable: workflow attribute display
  - name: attribute name. Note that both project related and activities's attribute can be displayed.
  - component: you can specify the component type associated with the attribute.

Possible values are

- text
- textarea
- wysiwyg
- date
- datetime
- select
- checkbox
- upload
- radiobox

The following screenshot illustrates all available types.



If you don't put any component type, a text component will be used.

- editable: boolean to set whether the value is editable

- The first form: a particular case

> **Note**
>
> *IMPORTANT*: By convention, the first form's state-name has to be an empty string.

Bonita workflows are actually instantiated after completing the first form. So this form does not match with any activities you've defined in the XPDL file, hence the tag has to be an empty string.

```
<form>
    <!-- The Start name is an empty String in the Bonita implementation -->
    <resource-bundle>request</resource-bundle>
    <delegated-view>true</delegated-view>
    <activity></activity>
    <variable name="start" component="date"/>
    <variable name="end" component="date"/>
    <!-- No need to specify Submit buttons in the Bonita implementation -->
</form>
```

The first form has also a special tag named "delegated-view". It basically determines whether the Process can be started from the Workflow User Portlet. If the value is "false", then the Process is considered as a system one. This means it is typically started behind the scene, by services in eXo. An example of such Process is "Content Validation". In that case, there is no need to specify variables nor a resource bundle, as no instantiation panel is displayed.

- **Submit** button

Bonita provides one action button: **Submit**. You don't have to specify it in the forms.xml. All attributes are instantiated with values submitted in the form when the user clicks on the "Submit" button. We'll see in the next section how to handle multiple choices with one button.

The forms.xml file used in this tutorial is available in the following :

- holiday-standalone.zip archive .

11.10. Bonita Resource bundle

11.11. Overview

You can associate an activity to a resource-bundle and customize the form display. You have to define a file .properties for each activity requiring a form.

11.12. Configuration

For example, in forms.xml the tag:

```

```

```
<resource-bundle>evaluation</resource-bundle>
```

refers to a file named "evaluation.properties" in src/conf directory.

* evaluation.properties file:

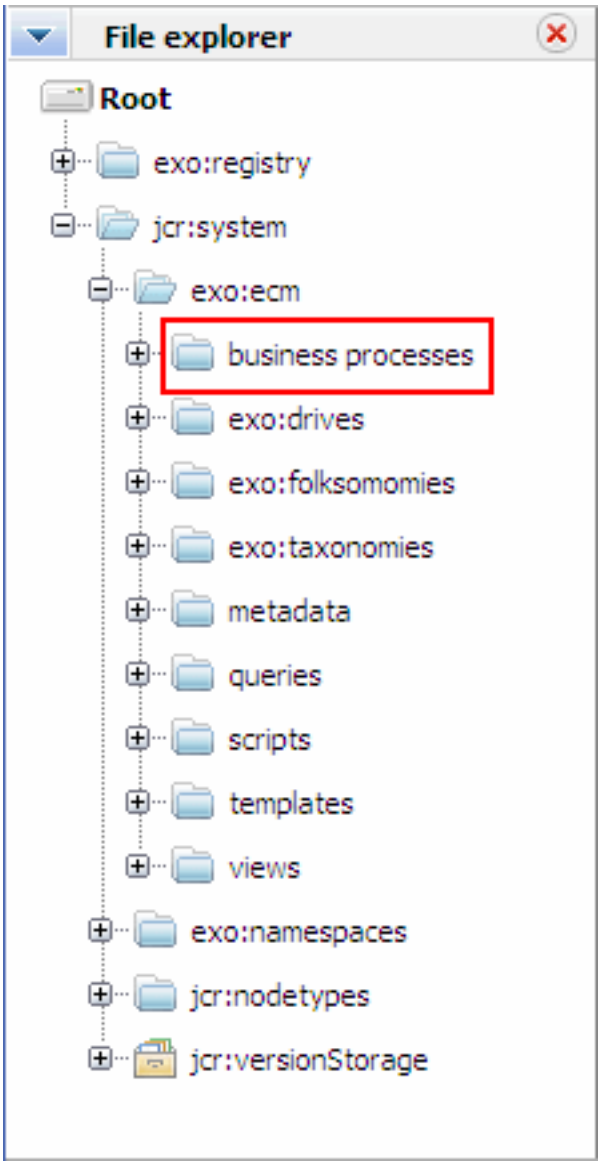
```
task-name=Evaluate holidays
title=Evaluate holidays

submit=Submit

#variable title and label
initiator.label=Initiator :
start.label=Start (dd/mm/yyyy) :
end.label=End (dd/mm/yyyy) :

Approve.submit=Approve
Approve.value=Approve
Disapprove.submit=Disapprove
Disapprove.value=Disapprove
Refuse.submit=Refuse
Refuse.value=Refuse
```

As we have seen, the value of the attribute can be set by the user in the Form or set programmatically by a hook.

11.13. Business Process in JCR

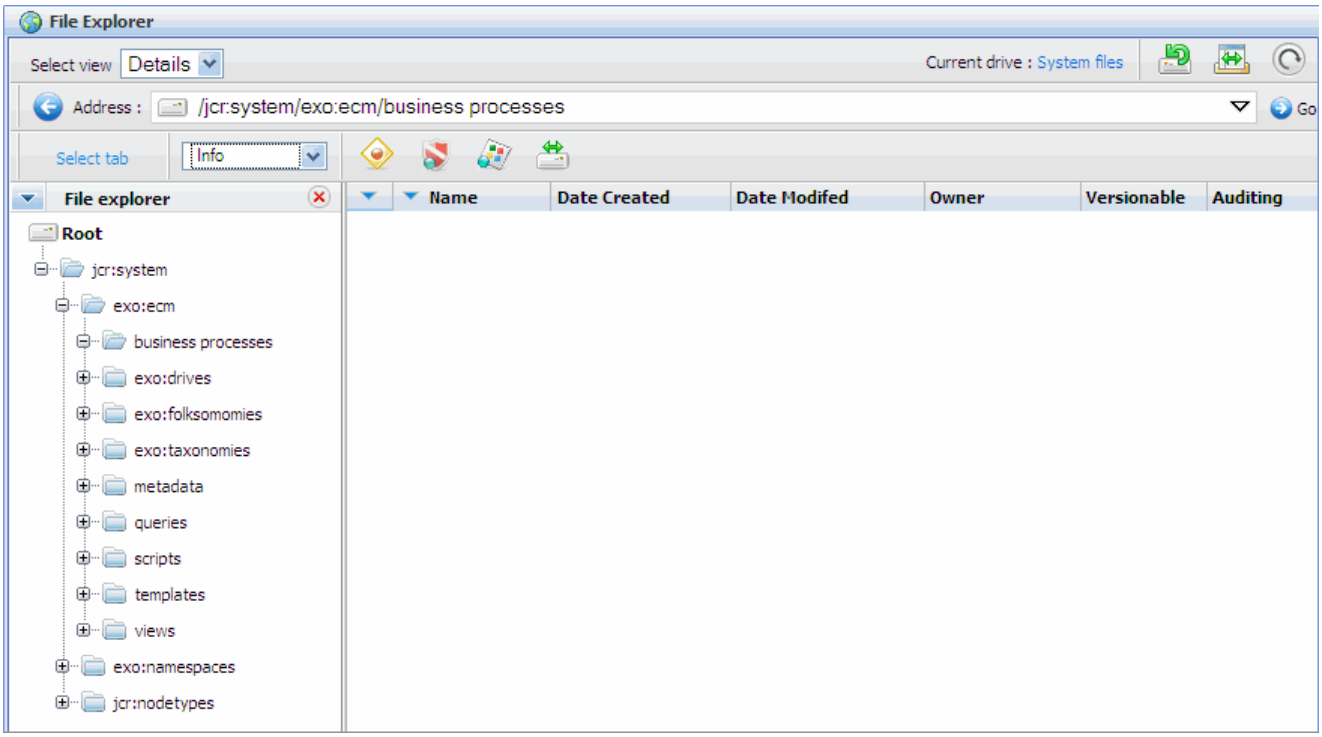
11.14. Overview

Every business process is deployed and stored automatically in the JCR and you can see it in File Explorer portlet. All the information can be found in the system workspace, in the "business processes" folder. The path is <code>jcr:system/exo:ecm/business processes</code>



11.15. Business Processes Files

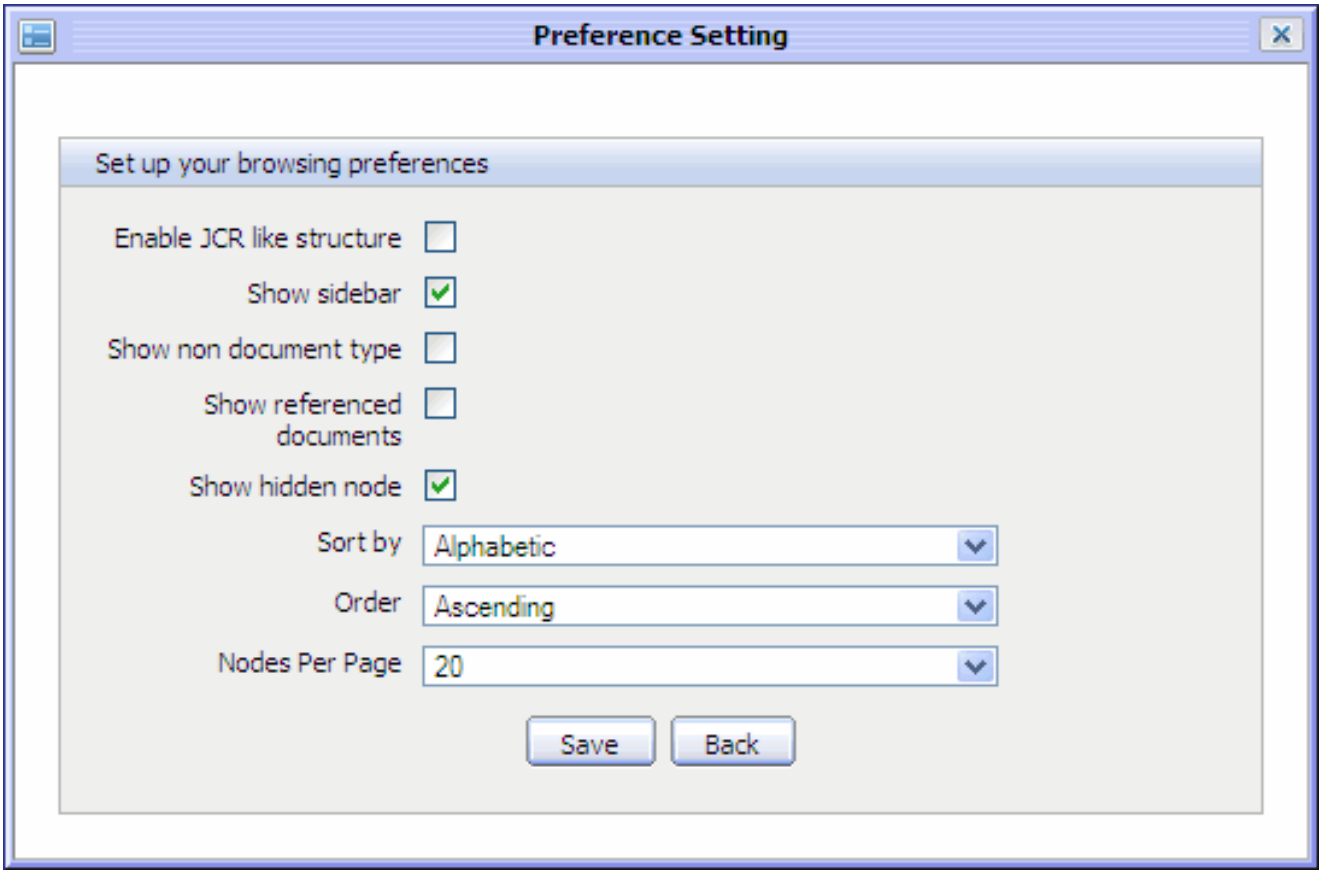
- By default the business processes are hidden. This is because the ECM Explorer only shows nodes which are considered as documents in the enterprise.



- You need to change the display preferences to see them.

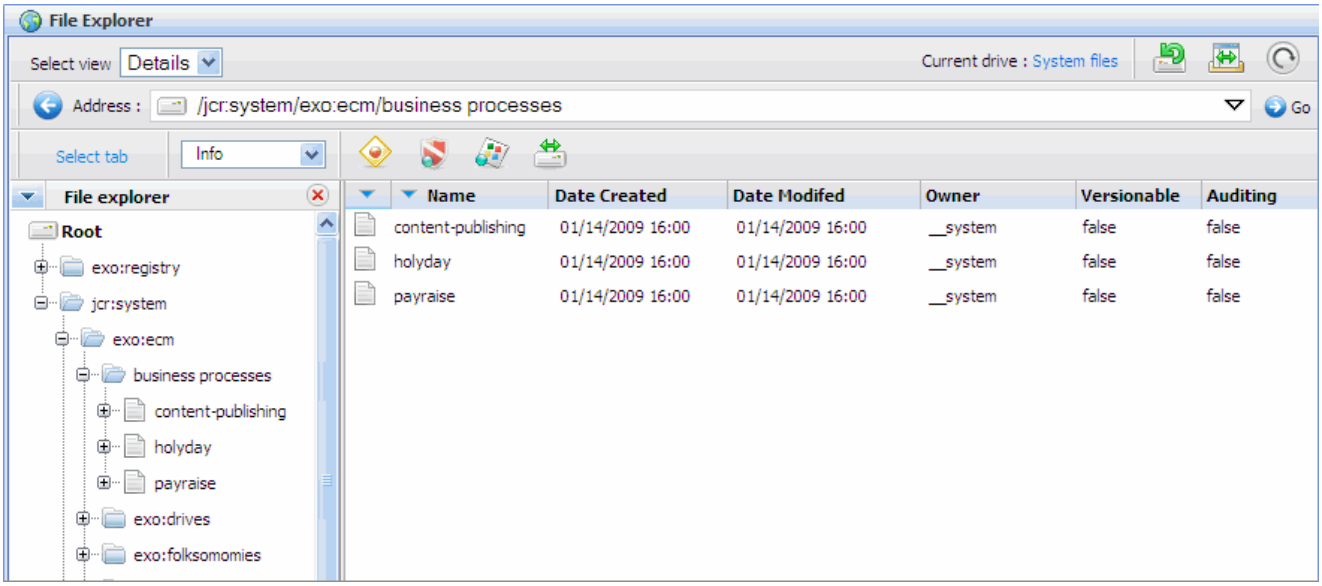
- Click on the icon

 . The preferences panel is displayed

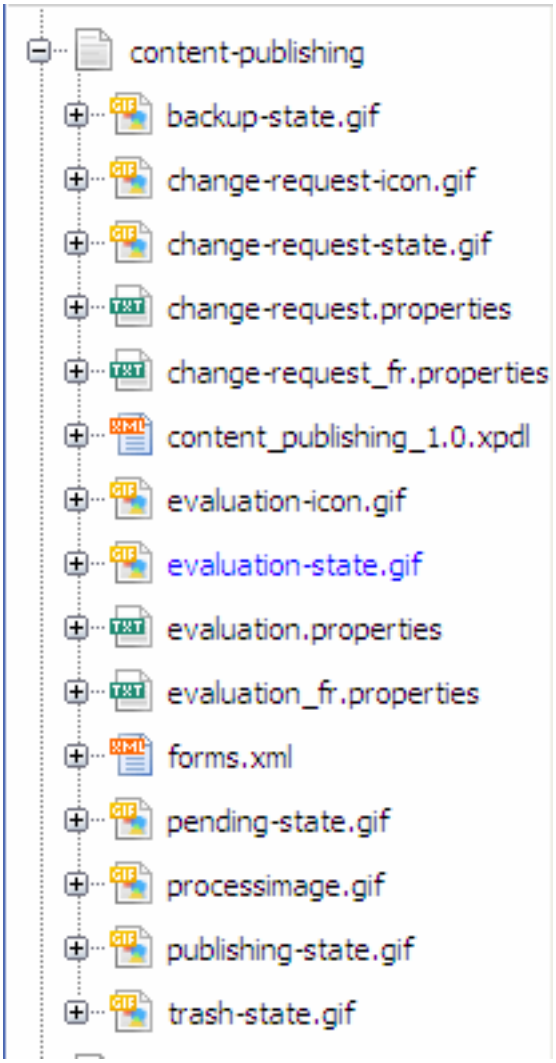


- Check "Show non document type" as above
- Click on **Save**

Now you can see business processes stored in the JCR.



- Click on any process to view files of this process. Example, view files of 'Content Publishing'



Chapter 12. Integration

12.1. Bonita Overview

eXo and Bonita team are working closely to deploy Bonita v4 - code name Nova Bonita - into eXo Platform Portal and WebOS.

Actually, this incoming Bonita version has its monitoring Console based on eXo WebOS:

<object width="425" height="344"><param name="movie" value="http://www.youtube.com/v/YV5qBblfCKw&hl=en&fs=1"></param><param name="allowFullScreen" value="true"></param><embed src="http://www.youtube.com/v/YV5qBblfCKw&hl=en&fs=1" type="application/x-shockwave-flash" allowfullscreen="true" width="425" height="344"></embed></object>

For the philosophy, please read this article, it is a bit old but the idea behind it is still true these days.

12.2. JBPM Configuration

12.3. Overview

Creating business processes that coordinate people, applications, and services, JBPM brings process automation to a much wider set of business problems, from embedded workflow to enterprise business process orchestration and JBPM.

Links:

- A tutorial about jBPM
- The jBPM documentation

12.4. Workflow Configuration

1. workflow is packaged as a jar file. We will take the content publishing workflow which is available in the default distribution of eXo ECM. This is the standard way of creating a workflow with JBPM. See the documentation for more information.

The following file define the process : **processdefinition.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<process-definition name="content-publishing">

  <description>
     This process validate document then public it, and backup it when publish time is expired
  </description>

  <!- SWIMLANES ->
  <swimlane name="initiator" />
  <!- START-STATE ->
  <start-state name="start">
```

```
    <task swimlane="initiator"/>
    <event type="node-leave">
      <action class="org.exoplatform.processes.publishing.InitialActionHandler"/>
    </event>
    <transition to="evaluation"/>
  </start-state>

  <!- NODES ->
  <task-node name="evaluation">
    <task name="evaluation" description="evaluate document">
      <assignment class="org.exoplatform.processes.publishing.ValidatorAssignmentHandler"/>
    </task>
    <transition name="approve" to="manage-publication"/>
    <transition name="disapprove" to="change-request"/>
    <transition name="delegate" to="delegate"/>
    <transition name="refuse" to="trash-movement"/>
  </task-node>

  <task-node name="change-request">
    <task name="change-request" description="update and request for validation again">
      <assignment class="org.exoplatform.processes.publishing.AuthorAssignmentHandler"/>
    </task>
    <transition name="submit" to="evaluation"/>
    <transition name="cancel" to="trash-movement"/>
  </task-node>

  <state name="trash-movement">
    <event type="node-enter">
      <action class="org.exoplatform.processes.publishing.TrashMovementActionHandler"/>
    </event>
    <transition name="move-done" to="end"/>
  </state>

  <process-state name="delegate">
    <event type="node-enter">
      <action class="org.exoplatform.processes.publishing.DelegateActionHandler"/>
    </event>
    <sub-process name="content-publishing" />
    <variable name="delegate_flg" access="read,write" mapped-name="delegate_flg" />
    <variable name="delegator" access="read,write" mapped-name="initiator" />
    <variable name="delegator" access="read,write" mapped-name="exo:validator" />
    <variable name="document-type" access="read,write" mapped-name="document-type" />
    <variable name="nodePath" access="read,write" mapped-name="nodePath" />
    <variable name="repository" access="read,write" mapped-name="repository" />
    <variable name="srcWorkspace" access="read,write" mapped-name="srcWorkspace" />
    <variable name="srcPath" access="read,write" mapped-name="srcPath" />
    <variable name="actionName" access="read,write" mapped-name="actionName" />
    <transition name="end" to="end"/>
  </process-state>

  <state name="manage-publication">
    <event type="node-enter">
      <action class="org.exoplatform.processes.publishing.SchedulePublicationTimerActionHandler"/>
    </event>
    <transition name="publication-done" to="manage-backup"/>
  </state>

  <state name="manage-backup">
    <event type="node-enter">
      <action class="org.exoplatform.processes.publishing.ScheduleBackupTimerActionHandler"/>
    </event>
    <transition name="backup-done" to="end"/>
  </state>

  <!- END-STATE ->
  <end-state name="end" />

</process-definition>
```

Here is the code of the **org.exoplatform.processes.publishing.ValidatorAssignmentHandler**:

```
/*
 * Copyright 2001-2003 The eXo platform SARL All rights reserved.
 * Please look at license.txt in info directory for more license detail.
 */
```

```
package org.exoplatform.processes.publishing;

import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.taskmgmt.def.AssignmentHandler;
import org.jbpm.taskmgmt.exe.Assignable;

/**
 * Created by the eXo platform team
 * User: Benjamin Mestrallet
 */
public class ValidatorAssignementHandler implements AssignmentHandler{

  public void assign(Assignable assignable, ExecutionContext executionContext) throws Exception {
    String validator = (String) executionContext.getVariable("exo:validator");
    assignable.setActorId(validator);
  }
}
```

Here is the code of **SchedulePublicationTimerActionHandler**.

```
/*
 * Created on Mar 24, 2005
 */
package org.exoplatform.processes.publishing;

import java.util.Date;

import org.jbpm.graph.def.Action;
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.instantiation.Delegation;
import org.jbpm.scheduler.exe.Timer;

/**
 * @author benjaminmestrallet
 */
public class SchedulePublicationTimerActionHandler extends MoveNodeActionHandler {

  private static final long serialVersionUID = 1L;

  public void execute(ExecutionContext context) {
    Date startDate = (Date) context.getVariable("startDate");
    Date currentDate = new Date();

    if (startDate.before(currentDate)) {
      try {
        moveNode(context);
      } catch (Exception e) {
        e.printStackTrace();
      }
      context.getToken().signal("move-done");
    } else {
      //Create and save the Action object
      Delegation delegation = new Delegation();
      delegation.setClassName("org.exoplatform.processes.contentvalidation.MoveNodeActionHandler");
      delegation.setProcessDefinition(context.getProcessDefinition());

      Action moveAction = new Action();
      moveAction.setName("moveAction");
      moveAction.setActionDelegation(delegation);
      context.getProcessDefinition().addAction(moveAction);

      //create the timer
      Timer timer = new Timer(context.getToken());
      timer.setName("publicationTimer");
      timer.setDueDate(startDate);
      timer.setGraphElement(context.getEventSource());
      timer.setTaskInstance(context.getTaskInstance());
      timer.setAction(moveAction);
      timer.setTransitionName("end");
      context.getSchedulerInstance().schedule(timer);
    }
  }

}
```

1 eXo Configuration

First, you need to create a mixin for the workflow :

```
<nodeType name="exo:workflowAction" isMixin="false" hasOrderableChildNodes="false" primaryItemName="">
  <supertypes>
    <supertype>exo:businessProcessAction</supertype>
  </supertypes>
  <propertyDefinitions>
    <propertyDefinition name="exo:validator" requiredType="String" autoCreated="false" mandatory="true"
      onParentVersion="COPY" protected="false" multiple="false"/>
    <propertyDefinition name="exo:businessProcess" requiredType="String" autoCreated="true" mandatory="true"
      onParentVersion="COPY" protected="true" multiple="false"> <!-- this property is used to set the name of
      <valueConstraints/>
      <defaultValues>
        <defaultValue>content-validation</defaultValue> <!-- Here you enter the name of the workflow -->
      </defaultValues>
    </propertyDefinition>
  </propertyDefinitions>
</nodeType>
```

To add your workflow to eXo you need to add this configuration to the portal.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>
  <component>
    <key>org.exoplatform.services.workflow.WorkflowServiceContainer</key>
    <type>org.exoplatform.services.workflow.impl.jbpm.WorkflowServiceContainerImpl</type>

    <component-plugins>
      <component-plugin>
        <name>deploy.predefined.processes</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.workflow.PredefinedProcessesPlugin</type>
        <init-params>
          <object-param>
            <name>predefined.processes</name>
            <description>load of default business processes</description>
            <object type="org.exoplatform.services.workflow.ProcessesConfig">
              <field name="processLocation"><string>war:/conf/bp</string></field> <!- the location of the jars c
              <field name="predefinedProcess">  <!- the jars containing the workflow ->
                <collection type="java.util.HashSet">
                  <value><string>/exo.ecm.bp.jbpm.content.validation-2.0.jar</string></value>
                </collection>
              </field>
            </object>
          </object-param>
        </init-params>
      </component-plugin>
    </component-plugins>

    <init-params>
      <value-param>
        <name>hibernate.service</name>
        <description>if  this parameter is not set or the value is empty, null or default.
          The workflow service will use the default Hibernate service.  If you modify this value,
          You need to update the external plugin as well</description>
        <value>default</value>
      </value-param>
    </init-params>
  </component>

</configuration>
```

The following configuration bind the workflow to a directory.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration>
  <external-component-plugins>
    <target-component>org.exoplatform.services.cms.actions.ActionServiceContainer</target-component>
      <component-plugin>
        <name>exo:businessProcessAction</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.cms.actions.impl.BPActionPlugin</type>
        <init-params>
          <object-param>

            <name>predefined.actions</name>

            <description>description</description>

            <object type="org.exoplatform.services.cms.actions.impl.ActionConfig">

              <field  name="repository"><string>repository</string></field>

              <field  name="workspace"><string>production</string></field>

              <field  name="actions">

                <collection type="java.util.ArrayList">

                  <value>

                    <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Action"> <!-- Init para

                      <field  name="type"><string>exo:workflowAction</string></field> <!-- - The name of the mix

                      <field  name="name"><string>publication</string></field>

                      <field  name="description"><string>publication workflow</string></field>

                      <field  name="srcWorkspace"><string>draft</string></field> <!-- Source Workspace: workspac

                      <field  name="srcPath"><string>/cms/publications</string></field> <!-- Source path: Node p

                      <field  name="lifecyclePhase"><string>add</string></field>            <!-- - Lifecycle Pha
(Add: Activate when have new node add to current node,
Remove: Activate when a node removed from current node,
Read: Activate when user click on custom action in this node)  -->

                      <field  name="roles"><string>*:/user</string></field>   <!-- Roles: declare permissions of

                      <field  name="variables"> <!-- init variable for the workflow instance -->
                        <string>exo:validator=member:/admin</string>
                      </field>

                      <field name="mixins"> <!-- Mixins type: Collections of ActionConfig$Mixin objects, that de
                        <collection type="java.util.ArrayList">
                          <value>
                            <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                              <field  name="name"><string>exo:move</string></field>
                              <field  name="properties">
                                <string>exo:destWorkspace=production;exo:destPath=/cms/publications;exo:reposito
                              </field>
                            </object>
                          </value>
                        </collection>
                      </field>

                    </object>
                  </value>
                </collection>
              </field>
            </object>
          </object-param>
        </init-params>
      </component-plugin>
 </external-component-plugins>



</configuration>
```