Nuxeo DAM 1.x

Developer Documentation

Table of Contents

Nuxeo DAM Developer documentation

Functional Overview

DAM stands for Digital Asset Management which means collection manager for digital assets which can be office (MS, OOo, PDF, ...) or multimedia (pictures, audio and video) files.

The main difference with Nuxeo DM (Document Management) is that DAM is not aimed at "producing" new documents with collaborative editing / review workflows but merely at browsing an existing collection that have been authored externally and imported in batch in the DAM application.

User will merely edit the properties / categories or add annotations to enrich the asset without editing the main attached file.

Download
Download this documentation in PDF.

Ajaxified search-centric UI

The navigation is not document-centric and organized in collaborative workspaces as in Nuxeo DM but is instead search-centric. Furthermore, the application ergonomy should feel like a rich collection browser such as iTunes or iPhoto by extensively using Ajax. Speed and responsiveness to user actions should be considered a major feature of Nuxeo DAM.

Ajax features are implemented using the Rich faces components and the Ajax4JSF extensions. The online demo is available here:

http://livedemo.exadel.com/richfaces-demo/richfaces/support.jsf?c=support&tab=usage

Some a4j tips are gathered on the following wiki page:

http://doc.nuxeo.org/xwiki/bin/view/Main/Ajax4JSF+Good+practices

In addition to JSFs ajax features, we plan to also reuse existing ajax features of Nuxeo DM such as the right click actions menu based on jquery plugins.

Development with Nuxeo DAM

Nuxeo DAM source files can be retrieved for development using Mercurial.

Nuxeo DAM Source code

The source code specific to the DAM project is located here:http://hg.nuxeo.org/nuxeo-dam/

To clone the Nuxeo DAM repository, use the following:

```
hg clone http://hg.nuxeo.org/nuxeo-dam
```

The main development branch is `develop`. To switch to this branch, use:

```
hg up develop
```

You can also update on stable branches, or releases tags.

Stable branches are named from the Nuxeo DAM version: `1.0`, `1.1`, ...

Release tags are named from the release version of Nuxeo DAM: `release-1.0`, `release-1.1`

Source code layout

The project layout is as follows:

- `nuxeo-dam-api`
  Common classes used in other modules (Constants, Exceptions, ...)
- `nuxeo-dam-core`
  Core document schemas definitions and other contributions to services
- `nuxeo-dam-importer`
  Contains the core importer classes to be used during the import process and JAX-RS classes to be able to use the HTTP importer.
- `nuxeo-dam-webapp-common`
  Common contributions and web resources needed by Nuxeo DAM and Nuxeo DM (when running Nuxeo DAM and Nuxeo DM on the same repository)
- `nuxeo-dam-webapp-core`
  DAM specific seam components
- `nuxeo-dam-webapp`
  Webapp resources for the DAM UI
- `nuxeo-dam-webapp-override`
  Contains the web resources that need to override the default ones copied in nuxeo.war.
- `nuxeo-dam-distribution`
  Assembly files to package the DAM application as:
  - A nuxeo.ear
  - A JBoss distribution
  - A Tomcat distribution

Continuous integration

The automated build / selenium test reports are hosted there:

http://qa.nuxeo.org/hudson/view/NX%20DAM/

Hudson also publishes nightly builds of JBoss and Tomcat distributions here: https://maven.nuxeo.org/nexus/content/groups/public-snapshot/org/nuxeo/dam/distribution/nuxeo-dam-distribution/1.1-SNAPSHOT/

Resources

- To help us prioritize new features, please see JIRA. You can vote on your favorite features.
- To ask questions and give feedback, please see the Nuxeo DAM Forum.

Additional Features

Core Features

The core features of Nuxeo DAM will be explained in this section:

- Nuxeo DAM Core
- Import
- Bulk Edit
- Video
  - Core module
  - Convert module
  - JSF module
  - Dependencies
- Audio

Nuxeo DAM Core

- Schema
- Document types
  - Existing Nuxeo DM document types
  - Nuxeo DAM document types
- Facets
- Physical document hierarchy layouting

Schema

A specific DAM schema is added: `dam_common`. It is used to store DAM related information like author of the assets, authoring date of the assets.

Document types

Existing Nuxeo DM document types

We reuse the document types defined in a standard Nuxeo, but types are overridden to use our custom schema:

- `Picture`: for image related files (jpeg, png, gif, bmp, ...)
- `File`: for Office document files (Microsoft Office documents, OpenOffice documents, PDF, text files, ...)
- `Video`: to be defined in a generic video module (like imaging one)
- `Audio`: to be defined in a generic audio module

Nuxeo DAM document types

2 new document types are defined for DAM:

- `ImportSetRoot`: the root folder where the `ImportSets` are stored. Extends the `Folder` type.
- `ImportSet`: extends the `Folder` type, contains the assets related to a given import.

One `ImportSet` is created for each import.

Facets

Nuxeo DAM defines a new facet `Asset` to specify which document types are assets.

This facet is added on the `Picture`, `File`, `Video` and `Audio` document types.

Physical document hierarchy layouting

As stated previously, an `ImportSet` document is created for each import, regardless of whether the file being imported is a composite file (eg. a zip file) or a single file (eg. a single jpg image).

They are all stored in the same `ImportSetRoot`: `/default-domain/import-sets` in the repository.

Each `ImportSet` document contains the assets imported.

Hierarchy example:

```
/default-domain/import-sets-root
                  |- folder1
                  |     |- import-set-1
                  |     |     |
                  |     |     |-file1
                  |     |     |-image1
                  |     |     |-subfolder1
                  |     |     |  |-image2
                  |     |     |  |-image3
                  |     |     `-audio_file1
                  |     `- import-set-2
                  |     |
                  |     |-file2
                  |     |-image4
                  |     `-audio_file2
                  `- folder2
                        |- import-set-3
                        |     |
                        |     |-file3
                        |     |-image5
                        |     |-subfolder2
                        |     |  |-image6
                        |     |  |-image7
                        |     `-audio_file3
                        `- import-set-4
                        |
                        |-file4
                        |-image8
                        `-audio_file4
```

The type of folder1 and folder2 is now "ImportFolder".

Import

Since Nuxeo DAM 1.1, the import process has technically changed. From a user point of view, it's still the same process.

This change allows us to provide an HTTP import of assets.

Outline of this document:

Import

To import assets through the web interface, you can follow the user guide: Import assets in Nuxeo DAM.

Technical overview

The import is now based on `nuxeo-platform-importer` module. All the classes related to the import in Nuxeo DAM are now in the `nuxeo-dam-importer` module.

- `nuxeo-dam-importer-core`
  Core classes used by the UI import and the HTTP import
- `nuxeo-dam-importer-jaxrs`

Contains only classes related to the HTTP import and defined all the JAX-RS modules to be able to launch a HTTP import.

Experimental stuff

The default behavior of the import is still the same one: you have to wait the end of the import process before being able to do anything on the UI.

Since the migration to `nuxeo-dam-import`, you can now set a new property `org.nuxeo.dam.import.async` to specify if you want a synchronous import, or an asynchronous one. To change it, go to the `nuxeo.conf` you use to launch Nuxeo DAM (pobably the one in `bin/` direc tory) and set the property to "`true`":

```
org.nuxeo.dam.import.async=true
```

RestartNuxeo DAM, and the next time you will do an import, you won't have to wait the end of the import after pushing "Create"! The import will be done asynchronously.

As this feature is experimental, a few more features still need to be added to the asynchronous import. These include:

- notification in the UI when an import you launched is finished
- notification by email that your import is finished (could be useful with long running imports)
- ability to see in the UI that an import is running

HTTP Import

To enable the HTTP import, you have to deploy the following jars:

- `nuxeo-dam-importer-jaxrs`
- `nuxeo-platform-importer-jaxrs`

You can also download a Nuxeo DAM distribution already with the HTTP import enabled:

- JBoss: https://maven.nuxeo.org/nexus/service/local/artifact/maven/redirect?r=public-releases&g=org.nuxeo.dam.distribution&a=nuxeo-dam-distribution&v=1.1&e=zip&c=jboss-importer
- Tomcat: https://maven.nuxeo.org/nexus/service/local/artifact/maven/redirect?r=public-releases&g=org.nuxeo.dam.distribution&a=nuxeo-dam-distribution&v=1.1&e=zip&c=tomcat-importer

Warning
This way to import is mostly useful for Administrators, or to fill Nuxeo DAM with assets before the first launch. It should not be accessible from outside.
We advise you to filter the following URLs through your Apache configuration:

- `/nuxeo/site/damImporter` (defined in `nuxeo-dam-importer-jaxrs` module)
- `/nuxeo/site/fileImporter` (defined in `nuxeo-platform-importer-jaxrs` module)
- `/nuxeo/site/randomImporter` (defined in `nuxeo-platform-importer-jaxrs` module)

How to use it

The assets must be stored on the same server as Nuxeo DAM, or accessible from the server.

Let's say we want to import all the assets stored in the folder `/tmp/to-import`, we can now use the following URL to import them in our running Nuxeo DAM:

```
http://localhost:8080/nuxeo/site/damImporter/run?inputPath=/tmp/to-import&importSetTit
le=My Import Set
```

If you are not logged in, you'll be asked to do it before the import starts.

From a shell (on the UNIX world), you can simply run:

```
curl -uAdministrator:Administrator
"http://localhost:8080/nuxeo/site/damImporter/run?inputPath=/tmp/to-import&importSetTi
tle=My Import Set"
```

List of parameters

Here are all the parameters you can use:

- `inputPath`: where the assets you want to import are stored.
- `importFolderPath`: the folder where you want to import the assets; if set, the importFolderTitle is not used.
- `importFolderTitle`: if you don't specify a folder path, you can set this parameter so that a folder will be created with the given title.
- `importSetTitle`: the title of your ImportSet. If not set, the title will default to something like "20100625 12:12", based on the date of import.

— Performance parameters

- `batchSize`: number of created documents to wait before doing an actual save on the database
- `nbThreads`: the max number of threads the importer can use
- `interactive`: if set to true, the call to the URL will block until the import is finished.

Metadata

The metadata can be automatically set on the assets during the import. To do that, you have to add a file named `metadata.properties` at the root of the folder you want to import. This file will contain all the metadata you want to set on the imported assets.

Sample file:

metadata.properties

```
dc\:title=Sample title on all assets
damc\:author=Peter
damc\:authoringDate=05/34/1980
dc\:subjects=art/cinema||art/architecture
```

⚠ Don't forget to escape the `:` character as it will be considered as a separator by the Java Properties class.

Here is the list of property types you can use within the `metadata.properties` file:

- `String`: just put your string as the value
- `Number`: just put your number
- `List`: separate your different values by `|`
- `Array`: separate your different values by `||`
- `Date`: the date format to use is: `MM/dd/yyyy`

You can put a new `metadata.properties` file in sub folders, it will override the metadata set by the parent `metadata.properties` file.

Write your own importer

To write your own importer, if you want something different than the default one, follow the documentation here: Nuxeo Bulk Document Importer.

Instead of using the `GenericMultiThreadedImporter`, you probably want to use the `DamMultiThreadedImporter` class to already have what is defined for Nuxeo DAM.

Bulk Edit

The Bulk Edit of Assets allows you to:

- Edit file descriptions, define metadata, and adjust settings for a selection of multiple assets
- Ease large-scale media file processing

How to change the Bulk Edit form

The form is based on a specific layout named `bulk_edit`. To change it you just need to override the default one by your own `bulk_edit` layout to display your custom metadata.

To do that, just add a new component with the following content:

```
<component name="com.sample.dam.layouts">
  <require>org.nuxeo.dam.layouts</require>

  <layout name="bulk_edit">
    <templates>
      <template mode="any">/layouts/layout_bulkedit_template.xhtml</template>
    </templates>
    <rows>
      <row>
        <widget>your_widget_1</widget>
      </row>
      <row>
        <widget>your_widget_2</widget>
      </row>
      <row>
        <widget>your_widget_3</widget>
      </row>
    </rows>
  </layout>
</component>
```

Be Careful
The widgets used in the `bulk_edit` layout should not have the property `required` set to `true`.
If the widgets you want to use have this property, redefine them in the layout without the `required` property.

Video

Nuxeo provides an addon with the following video features:

- In-browser video preview with JPEG thumbnails and quicktime player
- Storyboard extraction and time based navigation
- Darwin Streaming Server (a.k.a. DSS) integration for seeking forward large videos (using the storyboard for instance) without first buffering all the file locally.

The source code of this addon is located here: http://hg.nuxeo.org/addons/nuxeo-platform-video/

To clone it, use the following:

```
hg clone http://hg.nuxeo.org/addons/nuxeo-platform-video
```

Content of this section:

Core module

Video type definition

Schemas

- `video`: store the storyboarding items
- `streamable_media`: store the streamable version of a video file

Document Type

This module defined one document type `Video` which used the `video` and `streamable_media` schemas.

Core event listeners

VideoStoryboardListener

Compute the storyboard for document type that holds the `HasStoryboard` facet. The video storyboard is stored in the `vid:storyboard` field. Also update the `strm:duration` duration field.

VideoPreviewListener

Compute the a 2 thumbnails previews (same sizes as the picture previews) for documents that have the `HasVideoPreview` facet. The results is stored in the `picture` schema using the same picture adapter as the `Picture` documents. If the format is not supported by ffmpeg, a black thumbnail is generated. Also updates the `strm:duration` field.

MediaStreamingUpdaterListener

Asynchronous event listener to build the hinted streamable version of video to be used by the DSS_ integration. The results is stored in the `strm:streamable` field.

FileManagerService contribution

`VideoImporter` is contributed to the `FileManagerService` to create documents of type `Video` if the mimetype is matching when using the drag and drop plugin
or the DAM import button.

Convert module

This package holds the backend converters to compute JPEG thumbnails (preview and storyboard) and streamable version of videos for the DSS integration.

Provides contributions to the `CommandlineExecutorService`:

- get the ffmpeg output info (e.g. the duration in seconds) of a video file
- generating a single screenshot of a video file at a given time offset in seconds with ffmpeg
- generating a sequence of regularly spaces screenshots to compute the storyboard of a video file with ffmpeg
- converting a video from any format to mp4 (container format) H264 (video codec) + aac (audio codec) using handbrake (used for streaming)
- hinting mp4 files to make them suitable for streaming using DSS by using the MP4Box command.
- checking the presence of hinting tracks in a mp4 file using mp4creator to avoid recomputing them when not necessary (optim, not used yet).
- converting a video from any format to ogg (container format) theora (video codec) + vorbis (audio codec) using ffmpeg2theora (not used by default but could be use as a base for Icecast integration in the future as an alternative to DSS for instance).

All those `CommandlineExecutorService` contributions are wrapped into 3 higher level java classes that are contributed to the `ConversionService`:

- `ScreenshotConverter`: extract a single JPEG preview of the video
- `StoryboardConverter`: extract a list of JPEG files with time offset info
- `StreamableMediaConverter`: compute a streamable version of the video suitable for DSS_ integration.

JSF module

Provides basic JSF templates and backing seam components to be able to display a video player (using the quicktime plugin) that either use direct HTTP buffering or the RTSP-based URL that plays well with a darwin streaming server instance if a streamable version of the video is available.

This package also holds sample templates used in Nuxeo DAM to display a storyboard of a video that positions the Quicktime player to the right time offset when clicking on one of the thumbnails.

Dependencies

Mandatory

- ffmpeg is needed to compute JPEG previews, storyboard, and duration extraction: mandatory.

Mandatory if **DSS** mode enabled

- DSS: the streaming server itself.
- handbrake is used for encoding to h264/aac to compute the version streamable using darwin: only mandatory if the streaming server mode is enabled (disabled by default).
- MP4Box is used for track hinting for mp4 files: only mandatory if the streaming server mode is enabled (disabled by default).

Might be used in the future

- mp4creator will be used to avoid building streamable version of videos that are already streamable in their original version.
- ffmpeg2theora is an optional dependency used by a converter that is not used by default in either Nuxeo DAM or Nuxeo DM.

Audio

Customizing Nuxeo DAM

This chapter presents you how to configure and customize your Nuxeo DAM.

Table of Contents

Metadata Customization

This page presents how to customize Nuxeo DAM.

Integration of Nuxeo DM and Nuxeo DAM

There are some differences between Nuxeo DAM and Nuxeo DM; that's why it's necessary to make 2 different modules:

- common: for modifications that don't depend on Nuxeo DAM ; module can be deployed on Nuxeo DM only.
- dam: specific Nuxeo DAM module.

Nuxeo DM

In this part, we put changes in the common module.

Add custom metadata

1. Add an XSD file to describe our new schema, something like `/OSGI-INF/my_metadata-metadata.xsd`:

/OSGI-INF/my_metadata-metadata.xsd

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.nuxeo.org/dam/schemas/graphicsdeisgner/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:nxs="http://www.nuxeo.org/dam/schemas/graphicsdeisgner/">

  <xs:simpleType name="stringList">
    <xs:list itemType="xs:string" />
  </xs:simpleType>

  <xs:element name="documentType" type="xs:string" />
  <xs:element name="documentUses" type="nxs:stringList" />
  <xs:element name="documentFormat" type="xs:string" />
  <xs:element name="nuxeoProduct" type="nxs:stringList" />
</xs:schema>
```

2. Add an `/OSGI-INF/my_schema-contrib.xml` file: it permits to contribute to the existing schemas.

/OSGI-INF/my_schema-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.common.schemas">

  <require>org.nuxeo.ecm.core.schema.TypeService</require>

  <extension target="org.nuxeo.ecm.core.schema.TypeService" point="schema">
    <schema name="graphic_designer_metadata" prefix="gdm"
src="schemas/graphic_design_metadata.xsd" />
  </extension>

</component>
```

3. To deploy correctly our contribution, we need to add it in `/META-INF/MANIFEST.MF`.

```
 ...
 Nuxeo-component: OSGI-INF/my_schema-contrib.xml
```

4. Add a `/OSGI-INF/my_core_types-contrib.xml` to add our metadate into core type.
   DAM gets some base core types: ImportSet, File, Picture, Video, Audio.

/OSGI-INF/my_core_types-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.common.core.types">
  <require>org.nuxeo.dam.schemas</require>

  <extension target="org.nuxeo.ecm.core.schema.TypeService"
    point="doctype">

    <!-- ... -->

    <doctype name="File" extends="Document">
      <schema name="common" />
      <schema name="dam_common" />
      <schema name="ip_rights" />
      <schema name="file" />
      <schema name="dublincore" />
      <schema name="uid" />
      <schema name="graphic_designer_metadata" />
      <facet name="Downloadable" />
      <facet name="Versionable" />
      <facet name="Commentable" />
      <facet name="Asset" />
    </doctype>

    <!-- ... -->

  </extension>
</component>
```

5. Add contribution to the manifest file:

```
Nuxeo-component: OSGI-INF/my_schema-contrib.xml,
 OSGI-INF/my_core_types-contrib.xml
```

Add new layouts

Here, we define how to layout the new core type, and which widgets to use with it.

1. First, we need to define widgets and when to render fields using them.

/OSGI-INF/my_widgets-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.demo.common.widgets">
  <require>org.nuxeo.ecm.platform.forms.layout.WebLayoutManager</require>
  <require>org.nuxeo.ecm.platform.forms.layouts.webapp.base</require>

  <extension target="org.nuxeo.ecm.platform.forms.layout.WebLayoutManager"
    point="widgets">

    <widget name="documentType" type="selectOneDirectory">
      <translated>true</translated>
      <fields>
        <field>gdm:documentType</field>
      </fields>
      <properties widgetMode="any">
        <property name="directoryName">documentType</property>
        <property name="localize">true</property>
      </properties>
      <properties widgetMode="edit">
        <property name="ordering">label</property>
        <property name="styleClass">dataInputText</property>
      </properties>
    </widget>

    <!-- ... -->
  </extension>
</component>
```

2. Next, define the layout itself and which widget to display.

/OSGI-INF/my_layouts-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.common.layouts">

  <extension target="org.nuxeo.ecm.platform.forms.layout.WebLayoutManager"
    point="layouts">
    <layout name="graphics_designer">
      <templates>
        <template mode="any">/layouts/layout_default_template.xhtml</template>
      </templates>

      <rows>
        <row>
          <widget>documentType</widget>
        </row>
        <row>
          <widget>documentUses</widget>
        </row>
        <row>
          <widget>documentFormat</widget>
        </row>
        <row>
          <widget>nuxeoProduct</widget>
        </row>
      </rows>
    </layout>

  </extension>

</component>
```

Directories

1. Directory file must be placed in "directories" folder in the main resources, `/directories/products.csv` for example, and should look like that:

```
"id","label","obsolete"
"product1","label.documentProducts.product1","0"
"product2","label.documentProducts.product2","0"
"product3","label.documentProducts.product3","0"
"product4","label.documentProducts.product4","0"
"product5","label.documentProducts.product5","0"
```

2. Then, we need to declare and contribute it:

/OSGI-INF/directories-contrib.xml

```
<?xml version="1.0"?>
<component name="org.nuxeo.dam.gd.common.directories">
  <require>org.nuxeo.ecm.directory.sql.SQLDirectoryFactory</require>

  <extension target="org.nuxeo.ecm.directory.sql.SQLDirectoryFactory"
    point="directories">

    <directory name="product">
      <schema>vocabulary</schema>
      <dataSource>java:/nxsqldirectory</dataSource>
      <cacheTimeout>3600</cacheTimeout>
      <cacheMaxSize>1000</cacheMaxSize>
      <table>product</table>
      <idField>id</idField>
      <dataFile>directories/product.csv</dataFile>
      <autoincrementIdField>false</autoincrementIdField>
      <createTablePolicy>on_missing_columns</createTablePolicy>
    </directory>

    <!-- ... -->
  </extension>
</component>
```

3. And finally, we add contribution to our manifest, like always 

```
OSGI-INF/directories-contrib.xml
```

ECM Types

So now, we have:

- schema with core type declared
- a layout with associated widgets.

We can now assemble them in the ecm type contribution and define which layout to use in which case.

Create a contribution named something like that: `dam-demo-ecm-types-contrib.xml`.

/OSGI-INF/my_ecm-types-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.common.ecm.types">

  <require>org.nuxeo.dam.types</require>

  <extension target="org.nuxeo.ecm.platform.types.TypeService" point="types">

    <type id="File">
      <label>File</label>
      <icon>/icons/file.gif</icon>
      <bigIcon>/icons/file_100.png</bigIcon>
      <category>SimpleDocument</category>
      <description>File.description</description>
      <default-view>view_documents</default-view>
      <layouts mode="any">
        <layout>heading</layout>
        <layout>dam_common</layout>
        <layout>file</layout>
      </layouts>
      <layouts mode="edit">
        <layout>heading</layout>
        <layout>dam_common</layout>
        <layout>graphics_designer</layout>
        <layout>file</layout>
        <layout>dublincore</layout>
      </layouts>
      <!-- all content already on summary page -->
      <layouts mode="view" />
    </type>

    <!-- ... -->
  </extension>
<component>
```

Nuxeo DAM specific

Because of the general layout difference or some contributions which hack base behaviors, we now need to make several changes. That's why, we are now working on the dam module.

ImportSets

ImportSet is one of the Nuxeo DM behaviours that we want to change. To do so, you need to override early changes to layout import using "importset_heading" instead of "heading".

Be Careful ...
... to require previously created bundles.

/OSGI-INF/my_dam-ecm-types-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.dam.ecm.types">

  <require>org.nuxeo.dam.gd.common.ecm.types</require>
  <require>org.nuxeo.dam.types</require>

  <extension target="org.nuxeo.ecm.platform.types.TypeService"
    point="types">

    <type id="ImportSet">
      <label>ImportSet</label>
      <icon>/icons/import_set.png</icon>
      <bigIcon>/icons/import_set_100.png</bigIcon>
      <description>ImportSet.description</description>
      <category>DAM</category>
      <default-view>view_documents</default-view>
      <subtypes>
        <type>Folder</type>
        <type>File</type>
        <type>Picture</type>
        <type>Video</type>
        <type>Audio</type>
      </subtypes>
      <layouts mode="any">
        <layout>importset_heading</layout>
        <layout>dam_common</layout>
        <layout>graphics_designer</layout>
      </layouts>
    </type>

  </extension>
</component>
```

Inherited properties

In Nuxeo DAM, inherited properties service is linked with an ImportSet.

When importing a new document (with a core type that contains the "Asset" facet), if both the imported document and ImportSet are containing the same schema, the service can copy the ImportSet information in the new document.

To do this, we only need to contribute to inherited component and specify the desired schema.

In fact, Nuxeo DAM is listening on `aboutcreate` event and searches an inherited service and similar schema.

The component should be like this:

/OSGI-INF/my_inherited-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="org.nuxeo.dam.gd.common.inherited.properties.contrib">
  <require>org.nuxeo.dam.inherited.properties.contrib</require>

  <extension target="org.nuxeo.dam.core.service.InheritedPropertiesService"
    point="inheritedProperties">

    <inheritedProperties schema="graphic_designer_metadata" allProperties="true" />

  </extension>
</component>
```

Widget changes

If you have a `selectManyDirectory` widget, you certainly may want to gain some vertical place in left panel. To do that, there is a css selector that changes "select" positions. You just need to specify a size of 2, like that:

/OSGI-INF/my_dam-widgets-contrib.xml

```
    <widget name="nuxeoProduct" type="selectManyDirectory">
      <translated>true</translated>
      <fields>
        <field>gdm:nuxeoProduct</field>
      </fields>
      <properties widgetMode="any">
        <property name="directoryName">product</property>
        <property name="localize">true</property>
        <property name="size">2</property>
      </properties>
      <properties widgetMode="edit">
        <property name="ordering">label</property>
        <property name="cssStyleClass">dataInputText</property>
      </properties>
    </widget>
```

Modify/Add new Search Filters

1. First, we need to define a schema representing our query criteria:

/OSGI-INF/my_dam-schema-contrib.xml

```
<?xml version="1.0"?>
<xs:schema
  targetNamespace="http://www.nuxeo.com/ecm/demo/schemas/filterQuery/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:nxs="http://www.kisidam.org/ecm/schemas/filterQuery/">

  <xs:complexType name="stringList">
    <xs:sequence>
      <xs:element name="item" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="ecm_primaryType" type="nxs:stringList" />
  <xs:element name="ecm_fulltext" type="xs:string" />
  <xs:element name="dc_coverage" type="nxs:stringList" />
  <!-- ... -->
  <xs:element name="sortColumn" type="xs:string" default="dc:title" />
  <xs:element name="sortAscending" type="xs:boolean" default="true" />
</xs:schema>
```

2. After that, let's create a querymodel that matches our schema with predicates:

Don't forget to let FILTERED_DOCUMENTS as name to override default query Model.

/OSGI-INF/my-querymodel-contrib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="com.nuxeo.dam.demo.querymodels">

  <require>org.nuxeo.dam.querymodels</require>

  <extension
    target="org.nuxeo.ecm.core.search.api.client.querymodel.QueryModelService"
    point="model">

    <queryModel name="FILTERED_DOCUMENTS" docType="FilterQuery">
      <max>50</max>

      <whereClause>

        <predicate parameter="ecm:fulltext" operator="LIKE">
          <field schema="filter_query" name="ecm_fulltext" />
        </predicate>

        <predicate parameter="ecm:primaryType" operator="IN">
          <field schema="filter_query" name="ecm_primaryType" />
        </predicate>

        <!-- ... -->

        <fixedPart>
          ecm:isProxy = 0 AND ecm:mixinType !=
          'HiddenInNavigation' AND ecm:mixinType != 'Folderish' AND
          ecm:path STARTSWITH '/default-domain/import-root'
        </fixedPart>

      </whereClause>

      <sortColumn>
        <field schema="filter_query" name="sortColumn" />
      </sortColumn>

      <sortAscending>
        <field schema="filter_query" name="sortAscending" />
      </sortAscending>
    </queryModel>

  </extension>
</component>
```

3. And... add it to manifest.

```
OSGI-INF/my_querymodel-contrib.xml
```

4. At the end, just contribute to `search_filter.xhtml` to add the necessary query fields to be displayed. For a simple combobox, it should look like this:

/nuxeo.war/incl/search_filter.xhtml

```
<rich:panelMenuGroup
  label="#{messages['heading.filter.documentType']}"
  expanded="true" id="documentType">

  <nxdir:selectOneListbox
    value="#{filterActions.filterDocument.filter_query.documentType}"
    directoryName="documentType" id="documentTypeList"
    localize="true">
    <a4j:support event="onchange"
      reRender="filterResultTable,selectionView"
      action="#{filterActions.invalidateProvider}"
      eventsQueue="filterFormQueue" />
  </nxdir:selectOneListbox>
</rich:panelMenuGroup>
```

and for a simple text input field :

/nuxeo.war/incl/search_filter.xhtml

```
<rich:panelMenuGroup
  label="#{messages['heading.filter.fulltext']}" expanded="true"
  iconExpanded="none" id="keywords">

  <h:inputText id="fulltextFilter" class="searchInput"
    value="#{filterActions.filterDocument.filter_query.ecm_fulltext}">
    <a4j:support event="onkeyup"
      reRender="filterResultTable,selectionView"
      requestDelay="1000"
      action="#{filterActions.invalidateProvider}"
      ignoreDupResponses="true" eventsQueue="filterFormQueue" />
  </h:inputText>

</rich:panelMenuGroup>
```