Concept outdated - DAM 1.0 Asset Metadata

Purpose

DAM Module should support multiple Metadata definitions and in particular the Dublin Core Metadata .

Current implementation (Available Metadata in M5 Alpha 1)

Both dam module will have to be touched (dam and dam-app-asset).

Links

Dublin Core Specification

Nice concise overview of Dublin Core on Wikipedia

Implementation

Principles

The DAM API will provide a rich, but read-only access to Assets, their media and their metadata.
It is intended to be used by "clients" such as the STK.

On the other hand, the Assets App will not use this API and will access JCR directly, behaving much like the other Content apps.
The Assets App will not use this API currently as it was judged to be too much effort at this stage.

An Asset does not only support one metadata standard. An asset can support all metadata standards - though sometimes the user interface will only show the values of one standard.

Decisions

Currently only Magnolia and DublinCore metadata will be supported.
Eventually we will support many metadata standards.

Currently we will not support multiple values per property.
Eventually we will.

We will support the ability for customer to add custom Metadata fields to assets and access them in the templates.

Fields can be accessed via DublinCore field names.
By calling asset.getMetadata(SupportedMetaDataType.DUBLIN_CORE.name()). This returns a Metadata object where the DublinCore fields are exposed.

Roadmap

Additional things that we plan to implement in the future.

The system can extract metadata from media files and store in JCR.
The system can embed metadata in the files.
Metadata per asset type.

In templates/ftl you can access properties of a specific metadata type by its name ie : asset.dc.name)

Storage of Metadata in JCR

Fields will mostly be stored on the main Asset node as properties and multi-value properties.
Reasons: for simplicity, ease of use and speed of access.

More sophisticated metadata can be stored on subnodes to match the heirarchical structure of XMP and other metadata standards.

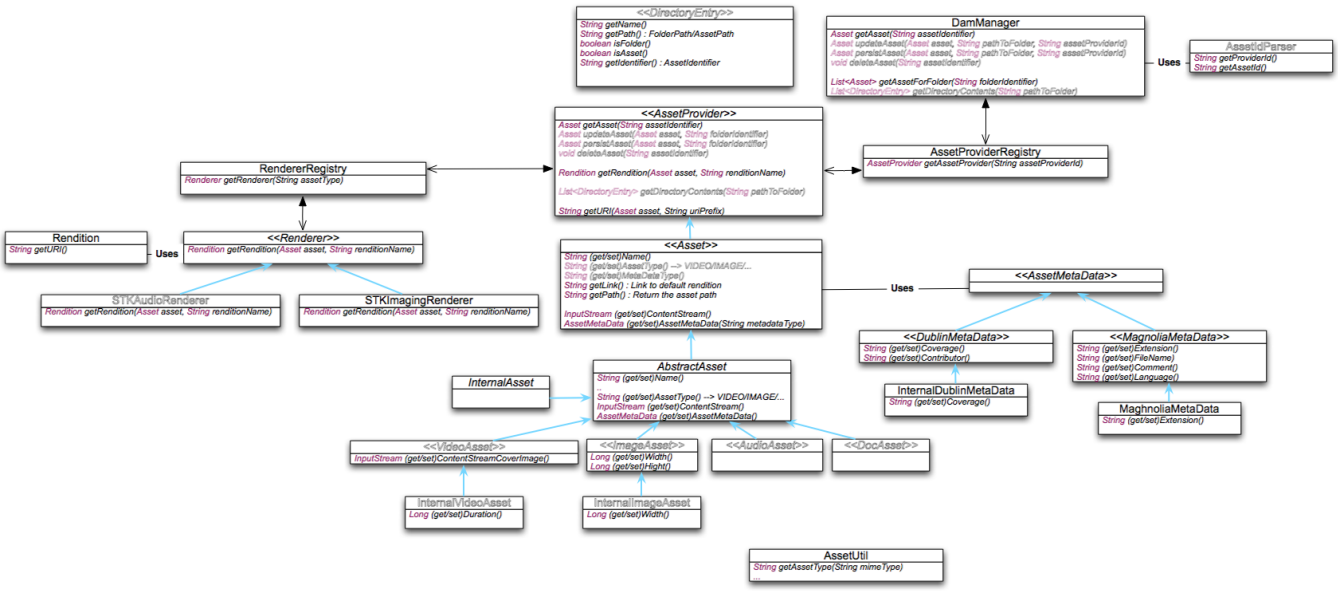
DAM Module API

Dam module has to give access to Metadata properties to templates.

To do so, the DAM API has to be revised in order to support multiple Metadata Types, and Templates has to have access to these Metadata properties in a easy way (in FTL, assetMap.description, asset.getMetadata("DUBLINSOMETHING").description). TODO: What is the real call here?

Big picture

During discussion and recent reviews we realized that the current implementation of the API has to be change in order to support this Metadata concept. It's the reason why so many classes are touched.



Mapping

Asset

Node	Property Name	Java Getter	Comments

mgnl:asset	language	Asset.getLanguage()	String representation of Local `http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt`
mgnl:asset	name	Asset.getName()	Name of the Asset.
mgnl:asset	identifier	Asset.getIdentifier()	Unique Identifier of the Asset. For an Internal asset this will be the node.getIdentifier() value.
mgnl:asset	title	Asset.getTitle()	Title of the Asset
mgnl:asset	type	Asset.getMediaType()	The mediaType is defined based on the mimeType. Currently this mediaType is defined as a String constant. Current Defined MediaType: **Audio / Video / Image / Document / Application**
mgnl:asset	subject	Asset.getSubject()	Subject of the Asset.
mgnl:asset	description	Asset.getDescription()	Description of the Asset.
mgnl:asset	caption	Asset.getCaption()	Caption of the Asset.
mgnl:asset	copyright	Asset.getCopyRight()	Copyright definition of the Asset.
mgnl:asset	mimeType	Asset.getMimeType()	Mime Type of the asset
jcr:content	size	Asset.getSize()	Asset File size.
jcr:content	jcr:data	Asset.getContentStream()	Asset Content Stream.
		Asset.getMetadata(): <<Metadata>>	Return the related Metadata.
		Asset.getLink():String	Return a String to the default rendition.
		Asset.getPath():String	Return a String to the Asset Path. For JCR this will be Node.getPath(), for a File Asset, this will be the absolute path to the Asset File

Metadata

MagnoliaMetadata

Node	Property Name	Java Getter	Comments
jcr:content	extension	AssetMetadata.getExtension()	
jcr:content	fileName	AssetMetadata.getFileName()	
jcr:content	jcr:mimeType	AssetMetadata.getMimeType()	

DublinMetadata

Node	Property Name	Java Getter	Comments
mgnl:asset	language	DublinMetadata.getLanguage()	String representation of Local `http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt`
mgnl:asset	title	DublinMetadata.getTitle()	Will be accessed by a cal to **Asset.getTitle()**
mgnl:asset	type	DublinMetadata.getType()	Will be accessed by a cal to **Asset.getType()**
mgnl:asset	subject	DublinMetadata.getSubject()	Will be accessed by a cal to **Asset.getSubject()**
mgnl:asset	description	DublinMetadata.getDescription()	Will be accessed by a cal to **Asset.getDescription()**

mgnl:asset	copyright	DublinMetadata.getRight()	Will be accessed by a cal to **Asset.getCopyRight()**
jcr:content	jcr:mimeType	DublinMetadata .getFormat()	Will be accessed by a cal to **Asset.getMimeType()**
mgnl:asset	jcr:identifier	DublinMetadata.getIdentifier()	
mgnl:asset	jcr:createdBy	DublinMetadata.getCreator()	
mgnl:asset	mgnl:lastModified	DublinMetadata.getDate()	
mgnl:asset	contributor	DublinMetadata.getContributor()	
mgnl:asset	coverage	DublinMetadata.getCoverage()	
mgnl:asset	publisher	DublinMetadata.getPublisher()	
mgnl:asset	relation	DublinMetadata.getRelation()	
mgnl:asset	source	DublinMetadata.getSource()	
mgnl:asset	MetadataType	Asset.getMetadataType()	Should return a Supported MetadataType (DUBLIN/IIPTC-IIM/EXIF). This info could be used to automatically associate the correct Metadata Type to the Asset and also by the Asset Dialogs to display the correct Metadata Tab/Fields

Questions

Should we introduce a configuration singleton used to define for the moment (should be later better done) the
Supported methadata types (DUBLIN/...)
Types (mage , InteractiveResource , MovingImage , ...) and mapping (mimeType --> type,....)

Review of the Ability of Templates to Access Metadata properties

How can customers access their custom fields from templates?

Notes from DAM team Meeting on January 31st>>>

How can customer add custom field?
Options:

1. Extend many classes
   asset.getMetadata("customMetadata").getCustomProperty()
   a. ⊖ Not really nice.
   b. ⊖ Not supported yet as we only support two metadata type.
   c. ⊕ Same mechanism for custom properties and build in properties
2. asset.getCustomProperty("myCustomProp") access a hashmap.
   a. ⊕ Use will have access to his custom property by using the standard DAM API.
   Asset Interface should expose this method, Asset provider should populate this hashMap
   b. ⊖ How to distinguish between custom properties and normal asset / Metadata properties
   (during the creation of this customProperty Map)
   c. ⊖ Non type safe.
   d. Is the property actually there? Return empty string otherwise.
3. Provide access to the node
   a. ⊖⊖⊖ Only works if asset is an InternalAsset.
4. Create an AssetMap object (Same pattern as ContentMap)
   This AssetMap should be created with an Asset and a HashMap (this HashMap contains all properties related to this Asset, even Metadatas)
   asset.fileName -> Return Asset.getFileName()
   asset.contributor --> Return Asset.getMetadata("dublinCore").getContributor()
   asset.myCustomPropertiy --> Will be on the responsibility of the AssetProvider to put this property in the HashMap
   a. ⊕ Easy syntax
   b. ⊕ Less Impact on the current implementation

    c. ⊖ Not able to distinguish in the syntax if a property is coming from the Asset, or from one of his Metadata.
5. Variant of 4 : Perform a Asset Metadata Mapping
   asset.dc_Contributor
   asset.fileName
    a. ⊕ Metadata are easily identified
    b. ⊖ AssetMap must be award of custom Metadata standard (All implemented MetadataStandard even custom).
    c. ⊖ AssetProvider has to have a Map of property name (dc_Contributor is link to JcrAssetNode.JcrContributorProperty value)
6. Ideal but How to:
   asset.metadata.dc.contributor
   asset.fileName
   asset.myCustomProperty
    a. ⊕ Better syntax
    b. ⊖ AssetMap must be award of custom Metadata standard
    c. ⊖ 2 level of Map. An AssetMap, referring to a MetadataMap

Answer:

1. **Start with 2:** The user will be able to access his custom fields using the Dam API.
2. **Implement 4** : Easiest solution for the user to access his custom properties in FTL's.
3. If time try to implement 6.

Related Tasks

type	key	summary	assignee	reporter	priority	status	resolution	created	updated	due

⚠ Unable to locate Jira server for this macro. It may be due to Application Link configuration.

Discussion

(Mostly moving comments and questions down here because we want them out of the way, but are not ready to delete them yet.)

Q: Should there be some kind of field mapping so I can access Metadata using normal DC field names?

A:

- For the fields that already exist, but have a different name, there should be a mapping to access those fields using the standard DC name.
- For example I should be able to request something like getDublinCore("creator"), or getDublinCore().getCreator() and it should give me the value for the jcr:createdBy field.

Q: How should the Metadata be represented in jcr?

- Additional fields on the node?
- A new mixin?
- A subnode with a new Metadata type?

A: We don't want to store it as a sub node as that will introduce the same performance problems as we had with mgnl:Metadata.

Proposals for storage of XMP compatible Metadata

Our Metadata "infrastructure" should be capable of handling multiple Metadata "views" on a flexible Metadata storage.
What is the best way to implement this, keeping performance, ease of use, standards-compliance, and reality into consideration.
(Reality: what standards specify vs. how people actually use it.)

- How can a client add their Metadata standard? (University departments, Government standards, Legal, Scientific)
- Must Metadata be searchable?
- Do we want to eventually support displaying a Metadata field as a column in list/tree views?

Metadata storage in JCR
Options:

- Blob: XMP Blob in one JCR property.
- Node Tree: Use a deep hierarchy of nodes to store exact hierarchy of XMP XML.
- Flat: All values stored flat on asset node - use property names to simulate heirarchical structure.
    - author-name
    - author-phone-cell
    - author-phone-home

	Blob	**Node Tree**	**Flat**
Pros	• Fully stores proper XMP	• Fully stores proper XMP	• Searchable • Sortable
Cons	• Hard to search • Hard/Impossible to sort	• Hard to search? • Hard to sort? • At least slower then "Flat".	• Probably hard to support full XMP • Need to decide what to support • Property names get long and complicated

Metadata object model
Options:

- Storage location (Options):
    - Store all values in Asset.
    - Store all values in MetadataObject
    - Store values across multiple MetadataObjects - one per standard type. (Maybe use harmonization rules - to store all possible in DublinCore, then store all possible in IPTC, etc)
- Storage technique (Options):
    - As properties with getters and setters
    - As a Hashmap - with text indices.
    - A heirarchical data structure
        - XML
        - TreeMap
- Working with data
    - Working with one Metadata standard i.e. DublinCore (Options):
        - An adapter with specific getters and setters.
          i.e. DublinCore.getDescription(asset);
        - A mapping - a Map of names of the properties in (A. the standard. B. our storage)
          i.e. asset.getMetadataValue(dublinCoreMap("description"));
    - View all data:
        - Should be possible to iterate/display/operate on all of the Metadata - regardless of the Metadata standard it belongs to.

How will the UI operate?

Our forms are designed to operate on configured sets of JCR nodes. This may be impacted by our other choices.

- If we decided to have an object for each Metadata standard - how would we need to change the forms to operate on them?
- If we keep all Metadata as flat properties on the asset node - we can use our existing system.