Concept outdated - DAM API 1.0

- Introduction
- Scope and requirements
    - Querying
    - Enumeration
- Media types
- Renditions
- Metadata
- Internal provider
- Consequences on existing templates
- Design
    - API class diagram
    - Main Components
        - DamManager
        - AssetProvider
        - AssetProviderRegistry
        - Asset
        - MediaType
        - AssetRenderer
        - AssetMetadata
        - Internal Asset Provider
    - Internal node structure
    - Rules
        - Nodes
        - Changing Metadate Type associated to an Asset
    - Impact

Introduction

This concept describes the DAM API following discussions in January.

The decision that every asset no matter where it came from would have a node in the dam workspace is gone. We will now support referring to an asset using the id of its provider in combination with the provider-specific asset id.

The design is also changed to include the concept of an original. When a file is uploaded to the dam we create an asset and an original. The original contains the binary data uploaded and is never changed.

We are also expanding on the metadata support.

See also additional concepts on these topics:

Concept - DAM Workspace Structure

Concept outdated - DAM 1.0 Asset Metadata

Scope and requirements

The DAM API in 5.0 will be used primarily in templating, templates and models. It will be used to access assets and will essentially provide read access only.

Assets may be stored internally (into the dam workspace), but could also come from external storage.

We will only provide support for assets stored in the dam workspace in 5.0. So called internal assets.

Querying

Enumeration

In Addition we should also have the ability to access list of assets based on directories, or search for assets based on type/properties.

Media types

We will support different media types and have different feature sets for them. We use the mime type of the asset to determine its media type.

We've identified five different media types:

- Images
- Audio
- Video
- Documents
- Flash (anybody using Flash these days? 😒)
- possibly many other types e.g. PDF, Slideshows (not PP but things like http://www.nytimes.com/slideshow/2013/01/24/travel/36-marin-slide-show.html?ref=travel), video links (youtube…)

Q: Do we accept any uploaded file and fallback to a default behaviour for these?

Renditions

A rendition is a transformation of an asset. The rendition is provided by an AssetRender. Renderers are configured per media type.

Metadata

We will support multiple metadata formats/specifications/schemas.

This is outlined in Concept outdated - DAM 1.0 Asset Metadata

Internal provider

Assets stored in the JCR are called internal assets. The internal provider manages these assets.

How the provider stores the assets is described in Concept - DAM Workspace Structure

Consequences on existing templates

Templates using the DMS and especially those using the STK will be effected as they migrate to using the DAM.

Known changes are:

- The property FileSize is now Size.
- The property FileExtension is now Extension.

Design

The API should be primarily a set of well documented interfaces.

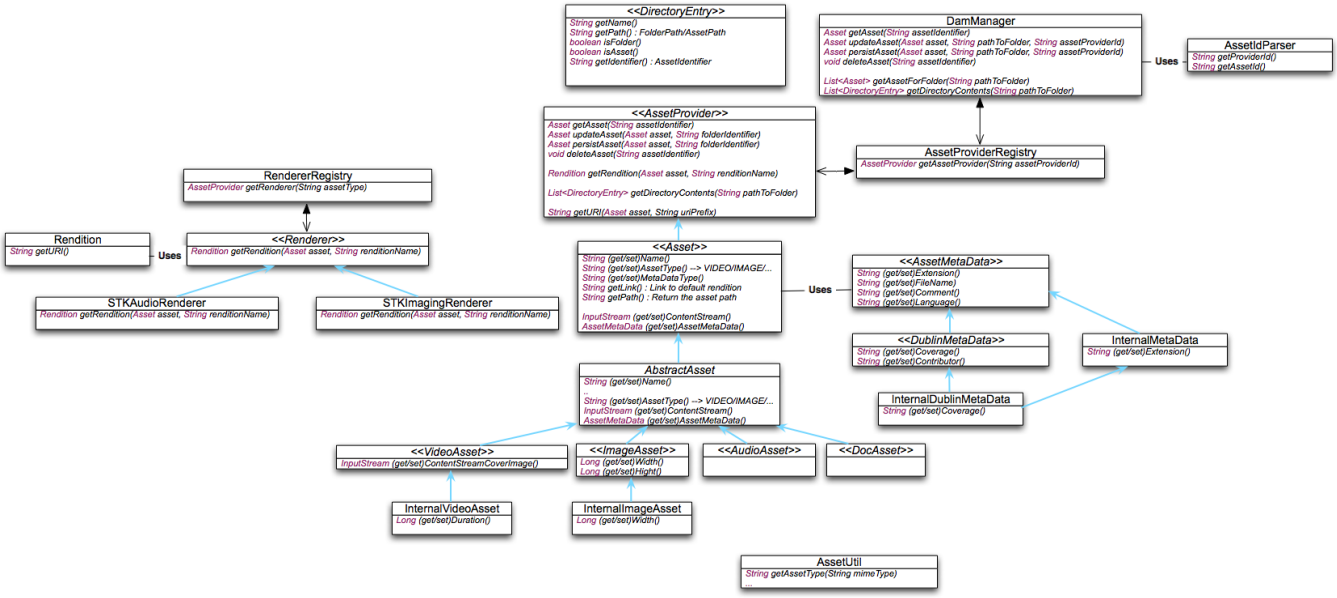
The API should not expose anything related to JCR.

An Asset is a simple POJO object containing all necessary getters and setters.

API class diagram

The API described in a class diagram.

> This diagram is out of date.



Main Components

DamManager

This is the entry point for third party.

Asset are retrieve based on

- id: concatenation of informations <<providerId>>:<<assetId>> (for an internal asset: dam:assetNodeIdentifier).
- an Asset object in addition with the provider id and directory path.

The DamManager uses the AssetProviderRegistry to define witch real implementation of AssetProvider to call in order to manipulate the Asset.

AssetProvider

The class implementing AssetProvider are responsible to physically handle the Asset:

- Access the physical storing sources and create/update/delete/modify an Asset.

AssetProviderRegistry

Asset

As already mentioned, Assets are simple POJO's with getter and setters. It's on the responsibility of the specific AssetProvider to correctly set/get the properties. Based on the assetMetadataType, the provider has to create the correct implementation of the specified Metadata.

MediaType

AssetRenderer

AssetMetadata

Based on the specified assetMetadataType, the specific Metadata implementation is created and linked to the Asset.
This gives us the ability to have an ImageAsset link to a default Metadata, and if specified by the Asset creator display (in the form), fulfill, and store DublinMetadata properties. So same AssetTypes could have different Metadata properties.

Internal Asset Provider

Internal node structure

Current DAM workspace hierarchy structure:

- `mgnl:folder` (nt:folder)
  - `mgnl:asset` (`nt:file`)
    `description`
    `title`
    `...`
    - `mgnl:resource` (`nt:resource`)
      `jcr:data` (`type="Binary"`)
      `size`
      `...`
  - `mgnl:asset` (`nt:file`
  - `...`

Rules

Nodes

`mgnl:asset`

Contains all Assets main properties, but also all Metadata properties that are not part of the AssetMetaData Interface definition.

`mgnl:resource`

Contains the Binary and all AssetMetaData properties.

Changing Metadate Type associated to an Asset

In this case, all the properties referring to the old Metadata type have to be removed.

Impact

Review the DAM migration task

Add an assetProviderName prefix to the properties referring to the DAM asset node.

Review the DAM UriMapping and Download Servlet.

Review STK interaction with DAM

⚠️Some current STK template functionalities may be harder to transpose. For example, the DownloadListModel allows via dialog to specify a path, file extension, to retrieve an Assets list. Currently done via JCR query.