Concept - DAM 1.0 future improvements

External AssetProviders

Currently we are only focused on the DAM AssetProviderType. The topic of additional AssetProviders is complicated. But we collect some thoughts here for now.

Brainstorming possible AssetProviderTypes

- Another workspace - ie "data" from Data module.
- Dropbox
- Flicker
- S3
- Akamai
- Google drive

What are the External provider use cases?

- I have a big collection of assets that I want to use on my site - i dont care where they are served, i just want to use them comfortably.
- I have a big collection of assets in a nice asset system that I will continue to use - I dont want to manage assets in magnolia's low-end dam system. But I would like to select assets from my system for use on pages.
- There are a few assets in another dam system that i want to use - i could import them.
- I want to serve some CDN hosted assets on my website.
    - I already have a bunch of CDN assets that I would like to select and use.
    - I would just like to have all of my website assets uploaded and served from a CDN. (Akamai and a CMS comments http://forums.adobe.com/message/4804400)
- I have a video on youtube, vimeo etc that I would like to serve.

Conclusions based on above use cases

1. It does not make sense to provide an asset from a CDN, because we create renditions of those and store those locally. A CDN implementation is something that should be applied afterwards.

DAMHandler & STK DAM Integration

- How does the UI look if I add additional DAMHandlers
    - Tabs for various options to upload images?

Problems

- Imaging module cannot work with an external asset on s3, flickr, akamai - if you want it served from there.
    - The point of these locations is that they store and serve the asset that you want to use. If you resize and store it locally, then that defeats the purpose.
    - I guess imaging module is still useful for adminCentral uses: thumbnails /

(Unimplemented) Concept for Configuration of AssetProviders

- DAM
    - config
        - providers
            - dam1
                - class = info.magnolia.dam.providers.dam
                - assetBrowserLabel = Dam
            - dropbox1
                - class = info.magnolia.dam.providers.dropbox
                - bucket = startrekmemories
                - authUserName = wil.wheaton
                - authKey = YR&#*YRHUHRU#HR#HR
                - assetBrowserLabel = Dropbox Wil.Wheaton

TODO

Further TODO

- Add upload to the Asset Browser.

- Create the Provider Registry
- Add provider options (from the registry) to the Asset Browser.
- Use Case: have two different dam workspace provider instances to the Asset Browser.
- Extra Credit: s3 or dropbox provider

DAM Concept Proposal (old and mostly implemented, here - for tracking)

The main conceptual changes from DAMSupport in 4.5 are:

- DAM/DAMSupport will not handle direct binary uploads to a content node. The default behaviour of all STK components will be to use assets in the dam. If a direct binary upload to a content node is desired, the "upload field" control can be used instead of a "dam field" control. (A new STK component would need to be created which contained this field.)
- A content node in a website will contain just one value to reference the DAM Asset, the identifier (UUID) of the Asset. Previously there were two values, one specifying the type of DAMHandlers, another containing either the identifier of the dam node, or the binary content. The DAM Asset node itself will store what type of assetProvider it uses.

New names:

- DAMHandlers => AssetProviders
- DAMSupport => DAM

Concept:

- There is a DAM module where all code is stored. (Except necessary STK integration)
- Content node has a field which is the identifier of the dam asset node.
- AssetNode has a field specifying assetProviderType.
- An AssetProvider registry enables the system to use the proper AssetProviderType for an Asset.
- Although assets could come from different sources, the asset node will have standard fields that will enable it to be displayed in lists etc.
- Nodes are of type mgnl:asset (nt:file) as this is the standard jcr type for dealing with binary assets. In the case that the node itself references a remote binary, the required jcr:content child node will be populated with minimum required content.
- Multi-Site:
    - AssetProviders are not aware of different sites.
    - On a multisite system, the sites can configure the root directory exposed by the AssetsApp. This way - each site could pull explicitly from a sub directory in the "dam" workspace.
        - The AssetsApp provides the AssetChooser which is opened from within the page editor - so this configuraration of the root directory for the assets app would also be reflected in the AssetChooser in the page editor.

Could have an externalImageProvider where you can put a url for any image - and it just imports it locally.

Notes

Asset Renditions

- Create a copy of an original
- Create a usage of a copy
- Show uses - all content items that use an asset.
- Behind the scenes
    - A mechanism that links usages, copies, originals
    - A mechanism that links an original and a copy
    - Usage must store its parameters
    - Copy should probably store parameters
- (Notes: Usages - do not show up in tree. Only Original and Copy.)

Questions

- I really cannot edit or replace an asset once it is used? Why not?
    - No. From Pascal, could lead to problems / inconsistencies. Please confirm

- Why can i revert to original - for a copy, which updates uses - but i cannot edit that copy once it has uses?
- Image Variations in the imaging module vs renditions

Answered Questions

- Originals and copies can be moved and named independently of each other. But they always retain the (hidden?) link.
- You can edit an original (does not automatically create a copy).

Purpose

The goal is to implement the ability to find assets and uses of assets as described in this UX specification: Digital asset management

This paper is focussed on the data structures necessary to achieve this and the strategies for finding the references, it does not cover the user interface.

Explicit Goals:

- With an Asset
  - Find an "Original" that it was derived from.
  - Find any "Copies" that were derived from it.
  - Find any "Uses" of the asset in another repository workspace, typically the website workspace.
  - Quickly find number of "Uses" of an asset so that it can be displayed in a column in list and tree views.

Some Use Cases: Scenarios for working with assets

Tracking uses/master/variant

Original / Copy

From a data modeling perspective, originals and copies have a one-to-many relationship. The most straightforward storage of the relationship is a foreign key on each copy. Therefore, every mgnl:asset node has an optional property "originalIdentifier" of type WEAKREFERENCE which stores the identifier of the asset that it was copied from.

We use a WEAKREFERENCE instead of a REFERENCE as it could be that a user wants to activate a copy but not its original to a public instance. (A REFERENCE would throw an error in this case because referential integrity would be broken, its referenced item would not be present.)

To find the original of a node, simply follow its originalIdentifier reference.
To find any copies of a node, search the dam workspace for all nodes which have the target nodes identifier as their originalIdentifier reference.

Uses

A "use" of an asset is any instance where that asset is used in another workspace, typically on a page or component in the website workspace. So a use is not another node in the dam workspace.

An asset has a one-to-many relationship with its uses. Each use stores the identifier of the asset. Unfortunately, there is not currently a specific property name for where the use stores the identifier.
To find the uses of an asset, search all properties of all nodes of the website (or potentially other workspace) workspace for the assets identifier.

Number of uses (Computed / Dependent / Cached Property)

In the UX mockups, in the list and tree views we see the number of uses of an asset. This might be slow to generate dynamically. A solution would be to store the number of uses of an asset on the asset node. The danger is that this could get out of sync with the actual uses. It is redundant data - just a "view" on the reference count. Is there a mechanism in JCR to store this kind of dependant, or computed property? If there is no built in way - we have to implement that such a value gets updated on an asset whenever the asset is added or removed from a component or page in the website workspace.

Questions

Should the originalIdentifier be optional, or should it be mandatory and just be empty on nodes with no original? There could be search performance advantages for one of the approaches.

The search for uses of an asset could be slow. The identifiers of the uses (a reverse reference) of an asset could also be "cached" on a subnode of the asset. I think we don't need to do this in the original implementation- but it could be an optimization if we find performance is unacceptable.