Introduction

The Hydra team has said from the word "go" that Hydra will use Fedora's Content Model Architecture (CMA) and to that extent will need a set of defined content models with associated 'Service Definition' and 'Service Deployment' objects (disseminators). It is our intention to share these here with the community. We are also making these pages available through the new 'content model architecture forum' elsewhere on this wiki. It became clear at the OR09 conference that content models are most useful if others coming to them understand the context in which they were developed - so watch for that too.

Hydra-provided content models, service definitions and service deployments will use Hydra's own namespace to distinguish them from other content that users may have, thus object PIDs will begin:

- hydra-cModel:
- hydra-sDef: or
- hydra-sDep:

General principles for constructing Hydra objects

All Hydra objects will subscribe to a metadata content model. The model will define several compulsory datastreams and several optional ones that would commonly be expected in a repository. Implementations may, of course, define additional metadata content models to supplement Hydra's list but we would recommend keeping a clean separation between metadata content models and content content models (sorry!).

Important note: It is not perhaps widely enough known that the size of Fedora's (FOXML) objects can have a significant effect on server performance. Hydra strongly recommends that all metadata datastreams other than RELS-EXT and Fedora's default DC should be of type 'managed' and not of type 'inline XML'.

We would expect all objects to understand the following disseminations:

- getDCMetadata
- getDescMetadata
- getRightsMetadata
- getContentMetadata

Content metadata may not be present in all objects. We have asked Fedora to change its code so that trying to access an optional datastream which is not present results, not in a 500 error, but in a clean response: likely an empty XML file.

Other disseminators under discussion:

- getOREMap - extracts or generates ORE map (need to decide how ORE map to be created)
- getCitation - extracts or generates citation (need to decide on format of citation and how created)
- getParents - returns a list of parent pid(s)
- getVerbs - return a full list of the verbs that the object understands through its disseminators

Additional functionality is added by having an object subscribe also to one or more local content models which support additional features. Thus, the basic Hydra content model for metadata provides for a MODS datastream. UK universities will additionally need a 'UKETD_DC' metadata datastream in objects that represent ETDs: this will be provided through an additional content model. A UK Fedora ETD object will subscribe to both.

Hydra objects that have content (ie are not metadata-only) will then also subscribe to one or more content models that describe the structure of their content-bearing datastreams; it is recommended that these content models do not reference metadata datastreams.

Some readers may realise that it is possible in software terms to go and 'get' a datastream direct from a Fedora repository. So why use disseminators? Hydra uses disseminators to provide an abstraction layer between the Fedora repository and the technology stack above it. Thus an institution might change the structure of its objects during the evolution of a repository but because the stack uses, for instance, 'getDCMetadata' to access a Dublin Core record the application need know nothing of the change; all that is required is that the disseminator for 'getDCMetadata' be rewritten to reflect the new structure.

Hydra disseminators: terminology

We distinguish between simple and complex disseminators:

- simple disseminator - a disseminator that simply returns contents of a datastream; i.e., a simple abstraction: getDescMetadata returns contents of descMetadata datastream. This is different than just using default Fedora disseminator.
  - NOT http://fedora-host:8080/fedora/get/obj-pid/descMetadata

Information on these pages has been superseded by the content on Hydra objects, content models (cModels) and disseminators as of August 2010. Please update links accordingly.
the advantage of this approach is that it provides an additional layer of abstraction than just using default Fedora disseminator which hardcodes the datastreamID in the dissemination URL. Enhancements can be added to the simple disseminators at a future date by simply modifying the sDef/sDep objects. Simple disseminators do not require any external services so they are easier to distribute as a package.

- complex disseminator - a disseminator that requires an external service to accomplish the dissemination; e.g.,
  - perform xslt transformation converting MODS to OAI_DC
  - parse a METS structMap to get a list of explicit children objects
  - perform a transformation to generate an ORE map

Most of the disseminators Hydra has thus far outlined fall into the "simple" category, but there are a few that will be complex and the Hydra team will carefully consider the ramifications of this for Hydra distributions.

Hydra sets

There are two basic models for managing "Sets", our preferred name over "collections" or "folders".

- Explicit set relationships in which the set object contains an explicit listing of its set members
- Implicit set relationships in which the set object has no explicit listing but rather contains some rule(s) for identifying its set members

In all cases there must be a single object that represents the set itself in the repository, an object that defines and describes the set (in the abstract and/or for specific UI use) and provides a reference point (a PID) for creating object associations to the set. The various models described below concern the manner in which member objects are identified and managed.

There are many relationships that could be used to define a set (explicit or implicit) in RDF. Hydra will always use 'hasMember' or its converse 'isMemberOf' as appropriate (cf 'hasPart' and 'isPartOf' for aggregate objects); this does not preclude users working with other relationships. Hydra will reserve 'isMemberOfCollection' for use in the specific case of OAI-PMH harvesting sets.

Explicit sets

Parent object may designate members via "hasMember {childPID}" triples in RELS-EXT, or

Parent object may designate members via a METS structmap or similar mechanism

Explicit sets represent a useful approach when there is a one-time determination of a closed set.

Implicit sets

The set object for an implicit set has no itemised 'knowledge' of its set members but contains the information needed to retrieve them. This may take the form of a query against the repository Resource Index (where the members each contain an 'isMemberOf' assertion in RELS-EXT) or a more general query or search across the repository (find all photographs where the subject is Barack Obama), or some other rules-based selection. An extra datastream will be required in the set object to contain the query or rules necessary to retrieve the set membership information.

Content models and disseminators

Common metadata content model

As noted above, all Hydra objects will subscribe to a common metadata content model which provides for the types of metadata that all objects are likely to need.

Datastreams as follows:

- DC (compulsory) – The Fedora built-in minimal descriptive metadata, possibly derived automatically from the DescriptiveMetadata datastream below.
- RELS-EXT (compulsory)
  - hasContentModel
  - isMemberOf or isMemberOfCollection (as needed to create groups of ETDs by type, source, etc.)
  - etc
- descMetadata (XML) (compulsory)
  - 'Out of the box' Hydra will expect MODS but we have already shown that it is relatively straightforward to modify Hydra to work with other schemas here. Largely it requires changing the indexing XSLT and possibly the Solr properties file
- rightsMetadata (XML) (compulsory), may contain
  - PREMIS premisRights, or
  - METS rightsMD, or
  - MODS accessConditions, or
  - a locally defined rights metadata structure , or
  - etc - but there must be a machine actionable and/or human readable entry here
- contentMetadata (XML) (optional, however it should be present in all objects (simple,compound or parent) that can present a splash page containing onward links in order to provide structural detail for displaying them; may contain
  - METS FileSec
  - METS StructMap
  - ORE map
  - locally defined schema The schema here was developed by Stanford and is being adopted by the Hydra partners.
• etc
  • technicalMetadata (XML) (optional), may contain
    • PREMIS premisObject
    • type specific (e.g., MIX for images)
  • etc
  • provenanceMetadata (XML) (optional)
    • eg PREMIS premisEvents
  • sourceMetadata (XML) (optional)
    • eg METS sourceMD snippet? (only a wrapper to object-specific MD)

Disseminations:
  • getDC
  • getDescMetadata
  • getRightsMetadata
  • getContentMetadata
  • getTechnicalMetadata
  • getProvenanceMetadata
  • getSourceMetadata

Common metadata content model
Common metadata SDef
Common metadata SDep

Explicit set

An explicit set may declare its membership using 'hasMember' relationships in RELS-EXT or by other means. However this is done it should support a dissemination to return a list of its members:

Dissemination
  • getMembers

Explicit Set content model
This content model is currently a placeholder with no associated SDef or SDep
Explicit Set SDef
Explicit Set SDep

Implicit set

The members of an implicit set are by their nature undeclared. An implicit set object must have a datastream that contains the query necessary to retrieve the member list (a query against the resource index, or a more general query).

Datastream
  • memberRules (XML)

Disseminations:
  • getMemberRules
  • getMembers

Implicit Set content model
Implicit set SDef Note 'getMembers' temporarily performs the same function as 'getMemberRules'
Implicit set SDep

Aggregation object (aka complex object, atomistic object)

Hydra generally favours complex (atomistic) objects over compound (multi-datastream) objects unless the content in all datastreams is identical but for, say, MIME-type or screen resolution or else there is a requirement only for a single content datastream. This has implications for most object classes; for instance, we take the view that because some ETDs (electronic dissertations and theses) may necessarily be complex (more than one datastream, for example pdf + multimedia) then all ETD objects should be complex; a single datastream ETD is just a special case - an aggregation object with a single child.

This object does not contain any data-bearing datastreams and so potentially need subscribe only to the common metadata model. However, it will be useful that it should have a content model for identification purposes and there is a possibility that it should have additional disseminations:

Possible additional disseminations:
  • getChildren
  • getOREMap
The children of an aggregation object must declare their relationship to it using an 'isPartOf' entry in their RELS-EXT datastream.

Datastreams
Currently none

Disseminations
Currently none

**Generic aggregation parent content model** provided for the purpose of easily identifying such an object.

**Generic simple content model (a single content-bearing delivery datastream)**

As well as providing for objects entire in their own right, this content model will form the basis for many objects that are children of a parent (aggregation) object. In the case of child objects, the relationship will be declared as necessary in the RELS-EXT datastream.

The optional 'original' datastream provides for keeping a copy of the binary content in the form originally submitted. For example, a repository may receive a Word file but convert it to PDF to go into the 'content' datastream for delivery. The Word file could go into the 'original' datastream as the repository's reference copy.

Datastreams
- content - potentially any MIME type
- original (optional) - potentially any MIME type

Disseminations
- getContent
- getOriginal (but see text above)

*Note 2010-03-17: the CM, SDef and SDep here do not yet contain implementation of 'original'*

**Generic content Content Model**
**Generic content SDef**
**Generic content SDep**

**Generic image model**

An example of a justifiable compound (multi-content-datastream) object where datastreams differ only by screen resolution and/or MIME type

Datastreams
- thumbnail - thumbnail resolution of image (jpeg, gif, png)
- screen - screen size resolution of image (jpeg, jp2)
- max - max deliverable resolution of image (jpeg,jp2)
- master - archival master image (jpeg, jp2, tiff, xml)

**Generic image Content model**
**Generic image SDef**
**Generic image SDep**

Disseminations
- getThumbnail - returns thumbnail datastream
- getScreen - returns screen-sized datastream
- getMax - returns max datastream
- getMaster - returns master datastream

**jpeg 2000 (.jp2) image model**

Colleagues at UVa have contributed the following which was originally developed for work outside Hydra. Here we provide links to the appropriate pages on SourceForge so that readers will get the benefit of any updates. Note, therefore, that the content model, SDef and SDep pids do not begin with the normal Hydra namespace.

Datastreams
- source - JPEG 2000 image
JP2k content model, SDef and (SDep) - Note: apparently the ampersands in the SDep on the linked page are improperly escaped. They are correct here.

Disseminations

- `getMetadata` - returns JSON-encoded Djakota image metadata
- `getRegion` - returns resource region in specified content type
- `getImageView` - loads image in the djatoka IIImage Viewer