Fedora Content Models

See also: Fedora Metadata Storage Philosophy, Current Behavior Implementations

Content models and asset actions are defined at slightly different levels. Asset actions are defined to promote interoperability between institutions and between types of repositories. A content model represents a particular type of object that is useful for a particular Fedora repository. A content model ensures that objects within the repository behave consistently; it may or may not be useful at other institutions. There should be a simple mapping between asset actions and content models, but this mapping depends on the actual content models used in a repository. Even though there will be differences between content models, it is worthwhile to share them. It is likely that some content models can be synchronized between institutions.

General principles for content models

- Derivative image files will be managed by Fedora.
- Derivative audio/video files will not be managed by Fedora, only referenced (due to their large size and the need to be managed by a streaming server).
- The master copies of all files will be stored in HPSS. They will be referenced by the metadata and accessible through disseminators. There will not be datastreams that point to master files, because they would not behave in the way that normal datastreams behave.
- Whenever objects or files are referenced, all references will be via PURLs. This allows a level of abstration between the reference and the actual object, so that objects may be managed as needed while remaining accessible.
- Whenever possible, metadata will be stored in a METS document in a METADATA datastream. For more details, see the Fedora Metadata Storage Philosophy.
- As much functionality as possible will be put into XSL. This makes it relatively easy to make changes to the functionality, rather than requiring changes in compiled code and/or complex Fedora behavior mechanisms.
- Item-level objects will primarily be responsible for delivering the contents of the item in a way that applications can easily deal with. Rendering of items will typcially be performed by the collection object, since a rendering usually contains collection-specific information. In the case of the default disseminator (described below), the item-level objects will typically redirect to the collection object to perform the needed rendering.

Available content models

Standard disseminators apply to most objects.

Type-specific content models:

- Audio Content Model
- Collection Content Model
- Finding Aid Content Model
- Generic File Content Model
- Image Content Model
- Journal Content Model
- Multi-copy Document Content Model
- Oral History Content Model
- Paged Document Content Model
- Text Content Model
- Video Content Model

Resource Index relationships

All item-level objects will have a relationship in the Resource Index that ties the item to a collection object. It is possible for an item to be a member of multiple collections, with the PURL providing the indication of the "primary" collection. No other relationships are necessary for non-hierarchical items.

Notes about storage of media

We will let Fedora manage all of our datastreams unless there is a compelling reason to do otherwise. Managed content is the best way to take advantage of Fedora's versioning capabilities. Managed content also reduces the amount of work in tracking multiple files that represent the same item (page images and transcriptions, various sizes of photographs).

XML data will always be stored as managed content (M) rather than internal XML content (X). This will give us better long-term performance when making many changes to a document.

Master media files present a complication. In general, we will be unable to store master files directly in the repository, so they cannot be managed content. We will store master copies in HPSS and have metadata in Fedora that points to them. We are investigating methods (like SRB) to make this more transparent to Fedora.

Whenever possible, write your interfaces in terms of the disseminators instead of the datastreams. This will make the entire system more modular. Datastreams may change to suit the needs of the objects, but disseminators will remain relatively constant. In the future, it may be useful to write XACML policies keeping datastreams "private", but we have not resorted to this level of control yet.

Utility objects

We will use the "util" namespace in Fedora to store objects that can be used across collections. For example, util:DC-XSL contains XSL transforms that can be applied to DC records to produce simple results.

Dealing with non-digitized items

Sometimes, we want to store objects in the repository that have not been digitized. This can happen when:

- We have metadata available from an external source, as in the case of the Hohenberger EAD file. We want the objects to be searchable, but we do not have digitized versions of everything.
- The primary "object" of interest is text. Users are content to read the TEI transcription of the document, and don't need scanned pages.

There are two ways to handle cases like these:

1. Implement only disseminators the object truly supports, and force all code that uses an object to check for its capabilities.
2. Implement the "Null Object" pattern, creating disseminators that implement the same behavior methods as digitized objects, but don't actually do anything useful in response to these methods.

We will take the first approach. This has the disadvantage of making external applications/stylesheets responsible for checking that an object supports requested behavior, but there are several advantages over Null Objects:

- Rendering of objects is more flexible, as the external application/stylesheet can apply a consistent rendering across the collection. With a Null Object, the rendering would have to be very generic, as the object would not know anything about the visual style of the collection containing it.
- Changes to the visual display of a collection are made in a single place (the collection stylesheets), rather than spread across the collection stylesheets and multiple Null Object rendering systems.
- It is more obvious when something is implemented incorrectly. An ugly error page demands to be fixed, while a rendering that includes text like "image not available" may be missed in testing.
- Only having functional methods listed with an object makes a developer's job easier. With Null Objects, developers would see many methods listed with an object that perform no useful function.

There are exceptions to this approach. In the case of the Default disseminator, non-digitized objects will sometimes have to respond with "image not available" or the equivalent. There may be other cases in which we want to create cleaner code at the application level and introducing a Null Object is the best approach.