RESTful Fedora Proposal

Proposed REST API for Fedora Including a lightweight (XForms)Administrative Interface

This document was originally produced on June 19, 2007 by MediaShelf, LLC in collaboration with the Fedora Team and the RUBRIC project The original version of the document is available at http://www.yourmediashelf.com/reference/fedora/webservices/REST%20Proposal%20-%20v1.1.pdf

Overview

The Problem Space

Fedora was created to support a diversity of Digital Asset Management applications. Rather than attempting to create a general purpose graphical front-end for the system, Fedora's developers chose to rely heavily on web service APIs. This allows implementers to create user interfaces suited specifically to their applications. However, it also means that every project wishing to use Fedora must first create a client application that consumes the web service APIs. This requirement has proven to be one of the primary barriers to adoption of Fedora.

This document contains a proposed solution to that problem. This solution focuses on refining the web service APIs and creating a number of special default disseminators that expose a lightweight, configurable administrative interface for Fedora. These modifications will make it possible for new implementers of Fedora to create minimal custom user interfaces by simply modifying Fedora's default administrative views. Based on these default administrative views, they can also create their own custom disseminators to extend the administrative user interface.

Without restricting the flexibility of Fedora in any way, these modifications build on Fedora's existing architecture in order to ease the process of adoption. Further, because this approach relies solely on Fedora itself to track and expose custom user interfaces, it increases the opportunities for projects to share, modify and reuse aspects of their user interfaces.

What's wrong with the current API? Why does it need to be updated?

It is very easy to expose Java methods as web services. However, simply exposing a web service does not mean that it is intuitive and useful. As Fedora has matured, it's web APIs have grown in an ad-hoc fashion. This has resulted in a hefty list of methods whose complexity obscures the elegance of Fedora's underlying Architecture.

Why a RESTful Interface to solve this problem?

The fundamental concept in making web service interfaces RESTful is that almost every method we expose can be reduced to a CRUD (Create, Retrieve, Update, or Delete) action. Based on this observation, it is possible to expose an intuitive and easy to maintain REST interface by breaking up all of your methods into a number of separate *controllers*, each of which handles a set of CRUD operations for some aspect of the underlying model.

For the purposes of this document, we will define a *controller* as "something that receives an HTTP request, triggers an action within your model, and returns a response." Each controller in a RESTful interface gets its own URL. This same URL can be used to perform all CRUD operations because HTTP methods (PUT, GET, POST, DELETE) easily map to the different CRUD actions. The end result is an interface with URLs exposing CRUD controllers whose actions are triggered by HTTP methods or *verbs*. This is the ideal *RESTful interface*.

An added innovation, which further simplifies a RESTful interface, is to interpret file extensions on a REST URL as requests for a specific response format. This means that requests sent to a URL ending in .xml will receive XML responses, .html will receive (X)HTML responses, and so on for any response type you wish to support (RSS, MOBILE, ATOM, etc)

Comment from Chris:Wouldn't http content negotiation be considered more RESTful? I do have a bias toward indicating the desired format right in the URI, but I just wonder what a REST purist would say about it.

_Response from Matt: _That is the technically pure way to do it, but HTTP content negotiation is so rarely implemented correctly on the client side (or even understood) that it is easier and more reliable to support filetype extensions in the URL. You could support both as long as you give precedence to the filetype extension, allowing that to override any content type specified in the HTTP headers.

: What's the upside to updating the API?

Updating the REST API would make Fedora's web service interface simpler and more intuitive while also making it easier to see the elegance of Fedora's architecture_._ Making this elegant architecture more apparent in the API encourages developers to leverage Fedora's strengths when creating client applications and middleware.

A proper RESTful API would also make it possible to build XForms that operate directly on Fedora objects and datastreams without requiring any intermediary applications or middleware. This presents a wonderful opportunity to build a lightweight, configurable administrative interface directly into Fedora.

For example, calling this URL

[http://host (http://%5Bhost%5D:%5Bport%5D/fedora/objects/demo:1/DC/edit):port/fedora/objects/demo:1/DC/edit]

or

[http://host (http://%5Bhost%5D:%5Bport%5D/fedora/objects/demo:1/DC;edit):port/fedora/objects/demo:1/DC;edit]

will call up the editor for the DC datastream on demo:1. By default, this editor would look similar to the "Datastream" tab in Fedora's swing-based admin GUI, but could be overridden to provide a specialized form for editing that datastream. In the fashion of the upcoming *Content Model Dissemination Architecture* (CMDA), one could associate specialized forms with a content model object. For all objects subscribing to that content model, special forms will be displayed rather than the default form.

i.e. In objects associated with content model A, a special form will be returned for Datastreams whose dsId is B.

What are the roadblocks to updating the API?

It will not be possible to make these API updates available for Fedora 2.2. This is due to timeline constraints and due to the fact that Fedora 2.2 does not support the CMDA.

MediaShelf's Proposed Project to Address this Problem

- Create a complete REST API for Fedora
- Amend problems in the existing management API
- Provide a lightweight, configurable administrative interface for creating and editing Fedora objects
- Establish best practices for overriding and extending the administrative interface

RESTful API Controllers/Methods

From a RESTful perspective, each Fedora repository is a network of resources. The repository itself is a resource. Within it there are objects, datastreams, disseminators, and relationships. We use the REST API to Create, Retrieve, Update, and Destroy (CRUD) these resources.

This section contains a breakdown of all of the controllers included in the proposed REST interface. The controllers are categorized by the resource they expose.

Each controller is accompanied by a description of how it will respond to the four HTTP verbs. The pertinent parameters and their implications are enumerated as well.

Comment from Matt: I'm pretty sure that by using /fedora/objects, /fedora/disseminators, etc. in all of the URLs, we avoid creating conflicts with the existing REST APIs. Is this correct?

**

Objects

From the REST perspective, when we operate on an object, we are operating on the object itself and its properties. Datastreams and relationships, which actually belong to objects, are addressed separately.

In addition to CRUD operations, the following actions can be performed on objects:

- Search for Objects
- Retrieve history, profile, or method list for an object
- Export objects (FOXML, METS)

===

```
http://[host]:[port]/fedora
```

* /objects*h2.

PUT, POST: Create Object (auto-assign pid)

Parameters: *message body* pidNamespace format logMessage

GET: Find Objects

Parameters: resultFields maxResults query sessionToken

GET: Generate PIDs ??

Parameters: *nextPID numPIDs pidNamespace*

```
http://[host]:[port]/fedora
```

* /objects/pid*===

PUT: Create Object

Parameters: message body pidNamespace format logMessage

POST: Update Object

Parameters: *message body state label ownerID logMessage force*

_Comment from Matt: _Would it make sense to allow updates based on replacement FOXML? If not, then the HTTP message body can be ignored (or used in another way).

GET: Retrieve Object

Extensions: .xml .foxml .mets

Parameters: *export format context history profile methods asOfDateTime*

DELETE: Purge Object

Parameters: *force logMessage*

===

```
http://[host]:[port]/fedora
```

* /objects/pid;edit*=h1.

Retrieve editor view for the object identified by pid

PUT, POST, GET: Return "Editor" View

By default, this URL will return an XForm based on the object editor view in the Fedora Admin GUI. This default XForm will be stored as a datastream within the fedora:ADMIN-VIEWS object.

DELETE (return error)

Note _from Matt_

```
: using ;edit rather than /edit prevents conflicts with datastreams with dsid of "edit". However, it might
actually be convenient to use /edit. That way, administrators could override the default "edit" view by
creating a datastream with a dsId "edit"
```

**Datastreams*h1.

=

```
http://[host]:[port]/fedora
```

* /datastreams/pid OR /objects/pid/datastreams*h2.

PUT, POST: Create or Update Datastream (auto assign dsID)

Parameters: message body] dsLocation altIDs dsLAbel versionable MIMEType formatURI dsLocation controlGroup dsState checksumType checksum logMessage

GET: Retrieve of List all of an Object's Datastreams

Parameters: *asOfDateTime dsState list*

```
http://[host]:[port]/fedora
```

* /datastreams/pid/dsId OR /objects/pid/dsid*===

PUT: Create or Update Datastream

This is equivalent to the existing addDatstream method in API-M, with one modification, which is that the client is allowed the option of submitting either valid XML or base64 encoded content directly within the message body of the HTTP request. Also, the value of *dsId* in the URL dictates the dsId for the new datastream.

Parameters: *message body dsLocation altIDs dsLAbel versionable MIMEType formatURI dsLocation controlGroup dsState checksumType checksum logMessage*

Comment from Matt: There is the standing issue of how data is passed into Fedora over HTTP. The leading options are multipart POST and base64 content in the message body. For now, I just put in base64 as the method of choice, but we should look further into this topic to decide on the best solution. Fundamentally, we want to ensure that the interface supports a wide variety of clients. Particularly, in order to support the XForms "edit" views, it must support submissions directly from XForms (which I think default to submitting base64 encoded content).

_Comment from Matt: _It would also be really great to support submission of local URIs using the file:// protocol.

POST: Update Datastream

Parameters: *message body**dsLocation altIDs dsLabel versionable MIMEType formatURI dsState checksumType checksum logMessage force*

GET: Retrieve Datastream

Returns the datastream.

Parameters: *compareChecksum asOfDateTime history*

DELETE: Purge Datastream

This is equivalent to the existing API-M method purgeDatastream.

Parameters: *startDT endDT logMessage force*

===

```
http://[host]:[port]/fedora
```

* /datastreams/pid/dsId;edit _OR _/objects/pid/dsId;edit*=h1.

Retrieve editor view for the datastream identified by dsId within the object identified by pid.

PUT, POST, GET: Return "Editor" View

By default, this URL will return an XForm similar to the datastream editor view in the Fedora Admin GUI. This default XForm will be stored as a datastream within the fedora:ADMIN-VIEWS object.

DELETE (return error)

**

**Disseminators*h1.

_Note: _This resource's URIs and parameters are likely to change once the CMDA is finalized.

=

```
http://[host]:[port]/fedora
```

* /disseminators/pid*===

PUT, POST: Create Disseminator (auto assign dissID)

Parameters: *message body bDefPid bMechPid dissLabel bindingMap dissState logMessage*

GET: Retrieve Disseminator

Parameters: *asOfDateTime dissState listMethods*

===

```
http://[host]:[port]/fedora
```

* /disseminators/pid/dissId*===

PUT: Create Disseminator

Parameters: *message body bDefPid bMechPid dissLabel bindingMap dissState logMessage*¬†¬†

POST: Update Disseminator

Parameters: *message body bMechPid dissLabel bindingMap dissState logMessage force*

GET: Retrieve Disseminator

Parameters: *history asOfDateTime*

DELETE: Purge Disseminator

Parameters: *force logMessage*

===

```
http://[host]:[port]/fedora
```

* /objects/pid/bdefPid/method*=h1.

GET: Retrieve Dissemination Results

Parameters: *asOfDateTime (method parameters*)

**

**Relationships*h1.

_Note: _This section will likely need to be rewritten to match with the CMDA.

One of the most powerful aspects of Fedora is its use of RDF to track relationships between objects. In the CMDA, these RDF relationships are also used to bind Objects (and content models) to disseminators.

=

```
http://[host]:[port]/fedora
```

* /relationships/pid*h2.

```
http://[host]:[port]/fedora
```

* /relationships/pid*===

PUT: Create Relationship

Parameters: *message body predicate target*

POST: Update Relationship ??

Parameters: message body ???

DELETE: Purge Relationship ??

Parameters: ???

Repository

The repository itself is represented as a resource but the actions that can be performed on the repository are limited to GET operations. In other words, you can use the REST API to get information about the repository, but you can't use it to modify the repository in any way.

===

```
http://[host]:[port]/fedora
```

* /repository*=h1.

GET: Get Repository Info

Parameters: *(none)*

**

**Users*h1.

Fedora was designed to rely on external applications (i.e. LDAP) to track users and groups. CRUD operations on users must be performed within those external user tracking applications. However, for convenience, the REST API includes operations that allow you to get information about users. Specifically, you can find out what information Fedora has access to regarding a user. This is primarily used to anticipate how Fedora's internal policy enforcement will be applied to the logged in user.

=

```
http://[host]:[port]/fedora
```

* /users/user*=h1.

GET: Get User Info

Parameters: *(none)*

**Amending Problems in the Existing Management API*h1.

The fundamental amendment to the existing management API will be to make it simpler. A RESTful API will allow those who consume the web service to think of their Fedora objects as resources. Simply by knowing the PID of their object and the DSID of any datastreams, clients will be able to easily access and operate on all aspects of those resources. In addition to making life easier for developers who create client applications for Fedora, this resource-oriented API will also make it possible to create disseminators that operate on Fedora objects.

This compact API also makes it easier for Fedora to gain new, intuitive features in the future. For example, it provides a great opportunity to add RSS /ATOM support natively into Fedora.

Another important amendment will be to implement proper support for HTTP uploads. Fedora is implemented in Java, but the web service APIs are meant to be consumed by clients written in any programming language. Data storage is one of the primary purposes for using Fedora. Thus, it should be simple to upload files directly into Fedora from any client, regardless of which language the client is implemented in (Java, XForms, PHP, Ruby, Flex, etc.). The upload servlet that is distributed with Fedora is meant to support this type of generic usage. However, in application, Fedora's reliance on the upload servlet often presents more of a barrier than a boon. This problem will be addressed the implementation of the new REST API.

**

**Lightweight, Configurable Administrative Interface*h1.

The most complicated and most promising aspect of this project will be the creation of a lightweight administrative interface built directly into Fedora. As described in the previous section, the interface will be bound to objects and datastreams as "edit" views. This interface will be provided by means of a combination of special Fedora objects, Fedora disseminators, and XForms. Customization of the interface will rely on the *Content Model Dissemination Architecture* (CMDA), which will be included in Fedora 3.0.

XForms

The default editor views distributed with Fedora will be implemented as XForms. They will be based on the Fedora-Admin GUI client. With their built-in support for web services, XML and XPath, XForms are a perfect solution for implementing lightweight user interfaces for Fedora.

_Comment from Matt: _Though it is not absolutely necessary, this is a key opportunity to survey users of the existing admin GUI and refine the interface to provide a more intuitive user experience.

Disseminators and Content Models

All Fedora objects automatically have an "Object Profile View", an "Item Index View", a "Dublin Core View" and a "Dissemination Index View". These views are provided by means of default disseminators. Likewise, retrieving a datastream is also achieved by means of a special disseminator (a datastream dissemination). The new administrative interface will build on this same pattern by relying on "Editor View" disseminations.

Unlike the existing default disseminators, the "Editor View" disseminators will provide opportunities for administrators to modify, replace, and override the views they return. These customizations will be achieved by either editing a fedora:ADMIN-VIEWS object or by overriding the "editor" views with special disseminators. The use of special disseminators to override editor views will rely on the new CMDA, which has an explicit notion of a *content model object*.

Fedora:ADMIN-VIEWS

The fedora:ADMIN-VIEWS object contains the default "editor" views for the repository. If no special editor view is associated with an object's content model, these default views are returned. The fedora:ADMIN-VIEWS object is entirely configurable, providing opportunities to override the forms exposed for any given object.

The default XForms views can be overridden in multiple ways. Administrators can modify any datastream in the fedora:ADMIN-VIEWS object, replacing it with a new form, or replacing it with a redirect to an external admin interface. Otherwise, administrators can override the defaults by associating alternate views with a content model, in which case those alternate views will be returned for instances of the content model.

Following the pattern of the fedora:ADMIN-VIEWS object, administrators can create any number of alternate "edit" views and bind them to objects or content models via disseminators. In this way, the default administrative user interface serves as a reference example for administrators and developers who wish to create custom editor views.

Issue

```
: Should there be the opportunity to return alternate views depending on user role? How could we implement this?
```

_Issue: _Should the distro versions of the view be stored outside of the datastore?

Issue: Should it always be possible to call the default edit view for an object or datastream, even if it has been overridden? i.e. If an administrator wants to see the whole object and edit at that level while librarians see a custom, restricted view.

**Establishing Best Practices*h1.

One of the key motivations of this project is to facilitate the creation of client applications and user interfaces that operate on Fedora. As a result, adequate documentation is equally important as the code itself. Developer documentation must be created for both the code and the API. Also, thorough documentation and tutorials must be created showing how to use and modify the administrative interface.

In this vein, the default "Edit View" XForms and their associated disseminators must be written to serve as reference examples. Developers should be able to look at these in order to learn how to create their own XForms disseminators or a variety of other custom views.

**

Open Topics==

Creating (and editing?) Disseminators

- Binding special edit views to specific objects
- Binding special edit views to instances of a content model
- Creating multiple edit views for a given object (or datastream?)
- Calling default "edit" view, even if it has been overridden,Ä¶

Controller Config

- Overriding default edit views
- Disabling edit views
- Using edit views that are outside of Fedora

Security

- Exposing "edit" views: does it pose a security risk?
- RESTful API: does it pose a security risk?
- Controlling access to edit views (XACML)
- Storing default edit views as datastreams in an editable object: potentially problematic?

Appendix I: Unresolved Issues in REST Mappings

1) (Resolved) Referring to all of an object's datastreams.

Applies to getDatastreams and addDatstream (to auto-assign dsId)

Solution 1

```
:
```

/objects/pid/datastreams

PUT (auto assign dsID)

GET (getDatstreams) ? list

Solution 2:

/datastreams/pid

PUT (auto assign dsID)

GET (getDatstreams) ? list

Conclusion

```
: Use both.
```

2) Referring to existing relationships (for update and purge)

3) Providing log messages when purging

HTTP message body vs. URL parameter

4) Modifying objects: Allow submission of new FOXML?

replace vs merge...

removing getNextPID and instead requiring creation of empty objects might necessitate supporting this...

5) Deprecate getNextPID? (instead make it easy to create empty objects with default FOXML and auto-assigned pid)

Would this make it necessary to allow HTTP POST of FOXML to /objects/pid?

6) Create HTML templates for all return types, allowing return of either HTML or XML for all methods?

7) When calling a dissemination, it would be nice to be able to ignore dissPID for special disseminators

8) Listing object methods

Use listMethods parameter?

9) Resuming a Search

Use a searchSession parameter?

10) Questionable URIs

The following operations have all been mapped to URIs, but those URIs might be inadequate in some way:

- Exporting Objects
- Getting next PID
- Getting Object XML
- Modifying Relationships
- Purging Relationships
- Listing Datastreams
- Listing Methods
- Resuming Searches

Appendix II: API-A and API-M Equivalents

An exhaustive listing of existing API-A and API-M methods and their REST equivalents is available for review. Please contact MediaShelf, LLC if you wish to receive a copy.

Portions of this document, including a number of comments from Chris Wilper of the Fedora team, have been taken from the entry in the Fedora Wiki titled Making_Fedora_RESTful. That wiki entry was originally authored by Matt Zumwalt from MediaShelf.