THE UNIVERSITY OF HULL

Provided without any warranty

RepoMMan Project

Deliverable D-D4

Development of Fedora materials

Richard Green

v1.0 2007-02-28

The RepoMMan Project

Project Director:	Ian Dolphin, Head of e-Strategy, University of Hull	(i.dolphin@hull.ac.uk)
Project Manager:	Richard Green	(r.green@hull.ac.uk)
Technical Lead:	Robert Sherratt	(r.sherratt@hull.ac.uk)
Repository Domain Specialist:	Chris Awre	(c.awre@hull.ac.uk)

The Repository Metadata and Management Project (RepoMMan) at the University of Hull is funded by the JISC Digital Repositories Programme. The project is being carried out by the University's e-Services Integration Group (e-SIG) within Academic Services.

Introduction

The RepoMMan project exists in large part to investigate the development of a workflow tool for the Fedora open-source repository software. At the time the project was conceived there was no available open-source client for Fedora, only an 'admin' client. Accordingly, Fedora 'out of the box' was a software tool with an associated, very steep learning curve and a user had to rely heavily on the documentation available on the Fedora website. As our experience developed we came to realise that the documentation appeared to lack some crucial elements and that, for a first time user, it was sometimes not easy to follow. Accordingly, the RepoMMan project team took a decision to document their development experience in the form of a supplementary Fedora manual in the hope that this would assist later adopters of the product. This is a step-by-step walk-through of our own experience and is offered as-is, without any sort of support or warranty.

This document is not intended to replace any of the Fedora documentation. Rather it is intended to be read with the Fedora materials in the hope that it will expand on some of the information there and help the reader avoid some of the pitfalls that we discovered. It may also provide a useful sequence through those materials during early work with Fedora.

This guide was written when Fedora 2.1 and 2.1.1 were the current versions. From version 2.2 (January 2007) setup has been very largely automated and new users should skip the first section. We believe that the great majority of what follows remains valid though it has not been systematically checked against the new version.

Installation and setup: Fedora 2.1(.1)

If you are installing version 2.1(.1), the Fedora software needs to be installed as per the 'Fedora Installation and Configuration Guide'. We had been working for some time with Fedora 2.0 and so we set up our initial installation to mimic 2.0's behaviour. This might also be a good strategy for those coming to Fedora 2.1 as their first experience of the product; leave the additional complexities of security until later.

The installation of the software is straightforward.

Assuming that you are going to start by experimenting with the supplied McKoi database, the first setup job is to initialise it (mckoi-init). We would strongly suggest that you do not alter any usernames or passwords at this stage: go with fedoraAdmin / fedoraAdmin.

Before you initialise the Fedora server itself you need to run the Fedora Setup Utility (fedora-setup). To make Fedora behave in a 2.0-like way, the command is:

```
fedora-setup no-ssl-authenticate-apim
```

This provides minimal security by restricting use of the Fedora Admin client (and API-M) to 'localhost' - the server itself. This can be altered later. Check carefully the response from your server to running this command. Make sure that there are no 'cannot copy' or 'cannot create' messages in it. If there are, the problem is hopefully the one below:

When setting up Fedora 2.1 on some machines running Windows XP the software fails to create two directories in the structure. The first is:

```
%FEDORA_HOME%\server\config
```

Create the empty directory by hand. The second is:

```
%FEDORA_HOME%\server\fedora-internal-use\fedora-internal-use-backend-service-
        policies
```

This latter may be a problem. Our eventual solution was to set up Fedora on a second XP machine (it works on by far the majority of XP machines, it seems!) as far as, and including, the fedora-setup stage, and copy the directory and its contents to the real server. This second installation was then deleted. The Fedora team are looking into this problem.

Once you have run fedora-setup successfully and have all the directories in place you should be able to start the database and successfully start the server without any fatal error messages. Remember, if you are using the McKoi database you will need to start the server with:

```
fedora-start mckoi
```

If all is well, you now need to do what you probably least want to do - stop the server again. There are a number of adjustments you need to make to the configuration file before you go on. So that you don't have to make them every time you choose a different security setting I suggest altering

```
%FEDORA_HOME%\fedora-internal-use\config\fedora-base.fcfg
```

- but not before you have made a safe copy of it!

Go through this base configuration file and make the changes suggested in 'Configuring the Fedora Server'. At this stage we would suggest altering a relatively few number of items:

- the adminEmailList
- the pidNamespace (keep this short if you can)
- retainPIDs (add your own)
- in the fedora.server.resourceIndex.ResourceIndexModule set 'level' to "2"

There are four occurrences of parameters that have a 'mckoivalue'. These allow the server to respond to a command like

```
fedora-start mckoi
```

The default value for each of these is the mysqlvalue. Our experience is that it is worth the effort to change value="......." on each of these lines to contain the mckoivalue. Then all you need type is

```
fedora-start
```

More importantly, other fedora commands that need an argument for the type of database will get the correct information from the configuration file without you needing to type it. (This saves some problems when you forget to type the argument, or didn't realise you needed one!)

Once you have made all these changes to fedora-base.fcfg, run setup again:

```
fedora-setup no-ssl-authenticate-apim
```

and a fedora.fcfg will be created with all your alterations applied. Now you should be able to restart the server.

Assuming all goes well, you might now want to ingest the demonstration objects. If you have made no changes other than the ones we suggested:

```
fedora-ingest-demos localhost 8080 fedoraAdmin fedoraAdmin http
```

You should be able to run the admin client on the server and check that all is working as it should. Before you can access the Fedora server from another machine you need to stop the server and alter one last thing in the fedora.fcfg file - the server's IP address in fedoraServerHost.

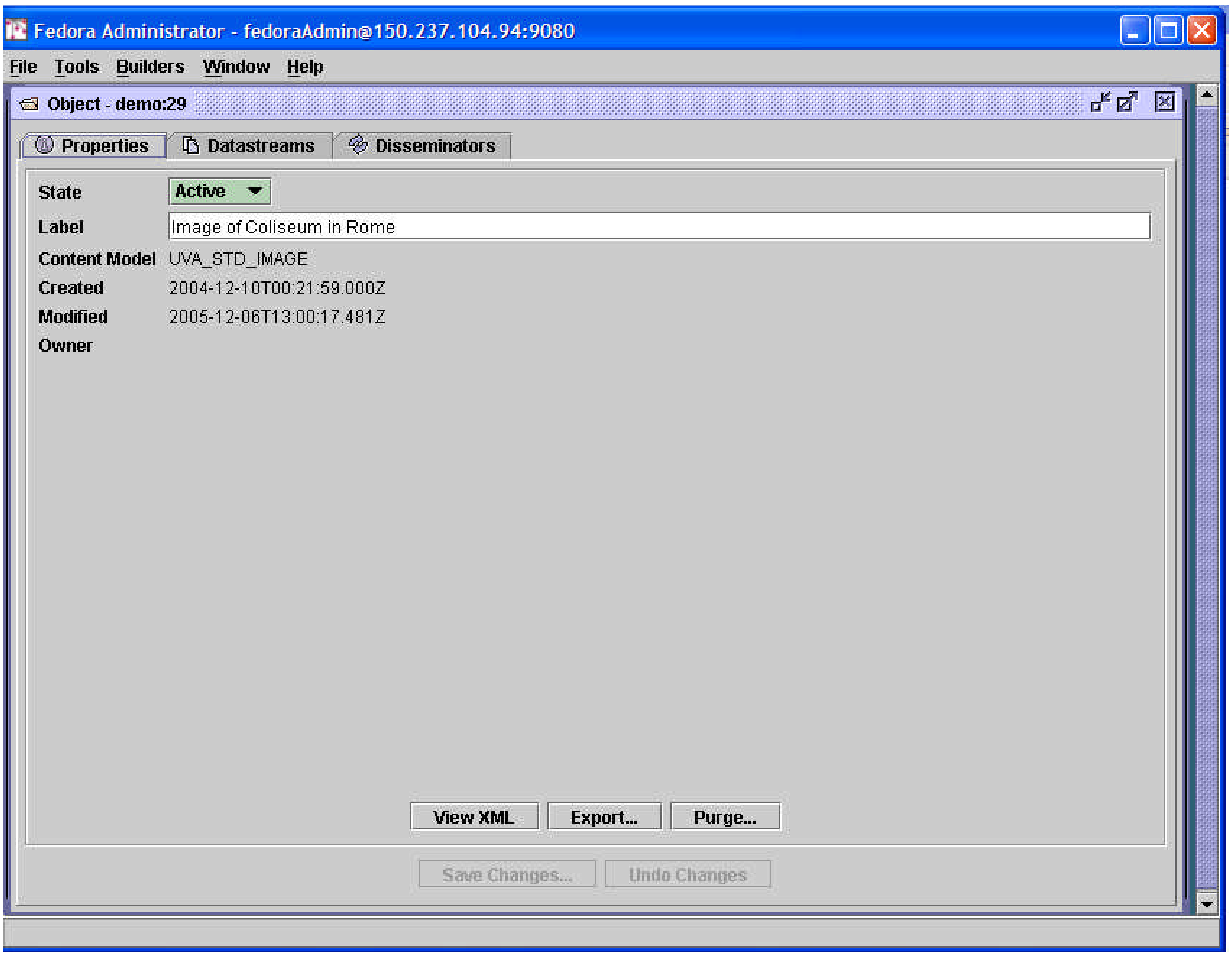
We would suggest working through all the early stages of this booklet with security left at this basic level and only change it when you get to the sections that deal with security. This will save you having to log into your server every time you call it up in a browser.

Fedora objects

The Fedora admin client provides two ways to open one of the demonstration objects.

- under the 'tools' menu, use the search routine which can find anything and everything, or
- from the 'file' menu choose 'open object'.

For our purposes here we need the object 'demo:29'. Either double click this on the list returned by the search tool (search for '*' - everything) or else enter the name in the input dialogue thrown up by 'open object'. Note that within the admin client object names are case sensitive. Using either of these methods you should end up with a Fedora object open in your admin client:



State	Active
Label	Image of Coliseum in Rome
Content Model	UVA_STD_IMAGE
Created	2004-12-10T00:21:59.000Z
Modified	2005-12-06T13:00:17.481Z
Owner	

'demo:29' is known as the object's persistent identifier or 'pid'. The representation of the object has three tabs across the top:

- Properties
- Datastreams, and
- Disseminators

Properties

The 'properties' tab tells us a little about the object:

- its 'state' - active, inactive or deleted
- a label - in this case 'Image of Coliseum in Rome'
- the content model that the object uses, if any - in this case, 'UVA_STD_IMG'
- the date it was created - 2004-12-10T00:21:59.000Z
- the date it was last modified - 2005-12-06T13:00:17.481Z, and
- its owner - this object doesn't claim to have one

The two date fields have the form yyyy-mm-dd followed by T (the time) hh:mm:ss.sss - the 'Z' (standing for 'zulu', a military notation) indicates Greenwich Mean Time.

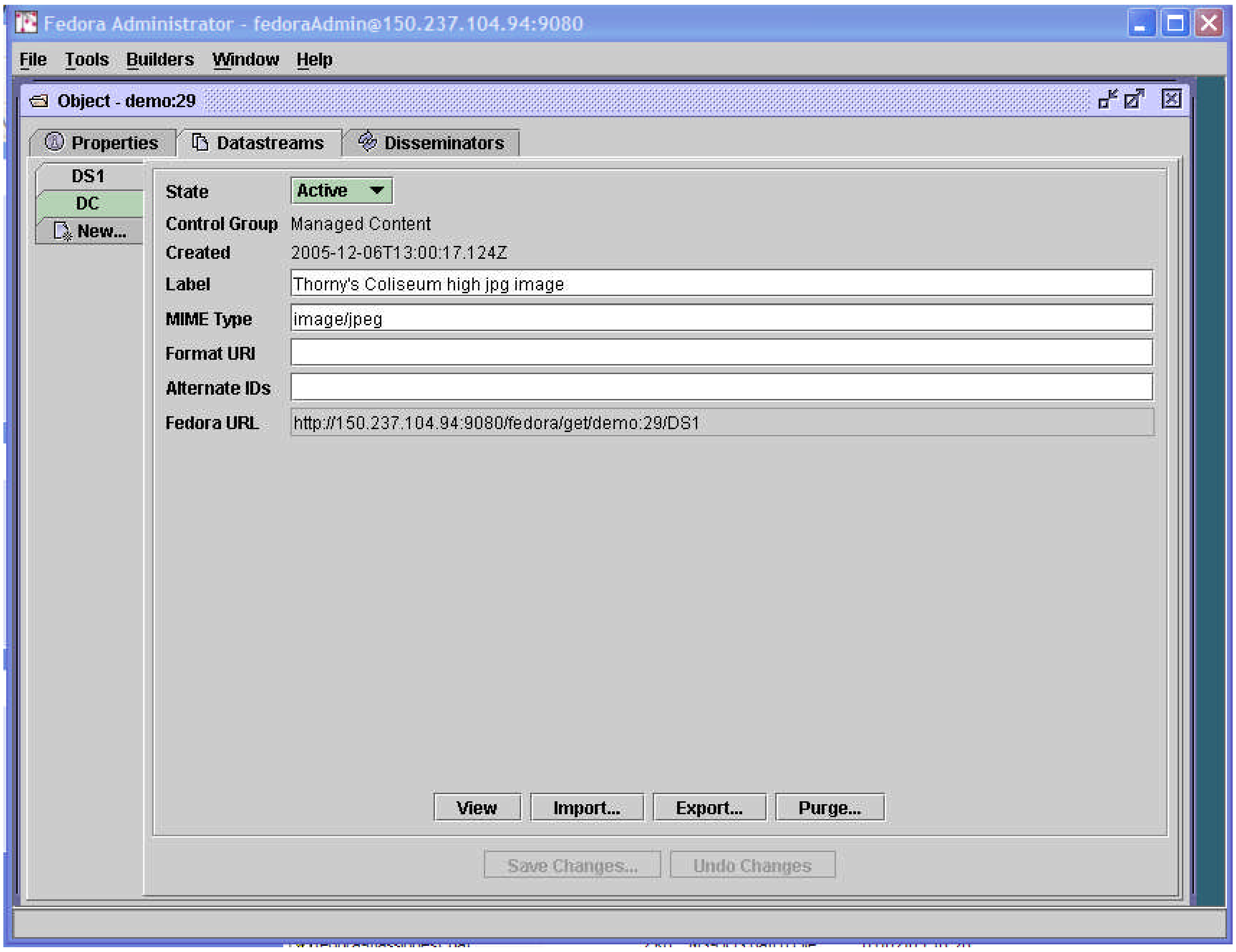
The buttons at the bottom

- View XML
- Export, and
- Purge

will be dealt with at a later time.

Datastreams

The 'datastreams' tab tells us something about the content of the object. In this case the object has two datastreams shown on tabs - DS1 and DC (and a tab for creating a new datastream).


Fedora Administrator - fedoraAdmin@150.237.104.94:9080

Object - demo:29

State: Active
Control Group: Managed Content
Created: 2005-12-06T13:00:17.124Z
Label: Thorny's Coliseum high jpg image
MIME Type: image/jpeg
Format URI:
Alternate IDs:
Fedora URL: http://150.237.104.94:9080/fedora/get/demo:29/DS1

Let's take these in the order they are shown.

The DS1 datastream is here what we might think of as the digital content of the object - in this case an image. 'DS1' is an arbitrary label that has been assigned to distinguish it from other datastreams and the name is probably not one that we would recommend - more on that when we come to creating objects.

There are several pieces of information here:

- its 'state' again
- a control group
- a 'created' date
- a label
- a MIME type
- a format URI, if any
- a list of alternate IDs, if any, and
- a Fedora URL

Let's take a quick look at some of these.

A datastream can belong to one of four control groups. It can be

- 'internal XML metadata' - this will be explained when we come to the 'DC' datastream next
- 'managed content' - this may be, for instance, an image file as here which is held within the repository. In fact the file can theoretically be of any type.
- 'externally referenced content' - this is content that is not held by the repository but elsewhere. It will be referenced by a URL and fetched from its host every time that the repository requests it.
- 'redirect' - here again the content is held elsewhere. 'Redirect' is used when the nature of the content is such that it must be delivered directly from the server on which it resides; an example of this would be streamed media

The datastream has a MIME type associated with it.

When it was created, Fedora assigned to the datastream a unique URL. If you enter this into a browser - 'Hey Presto,' there is the image associated with this datastream. If you doubt me, try it and compare the result with clicking the 'View' button at the bottom of the page in the admin client.

Let's now turn to the 'DC' datastream. This is automatically created by Fedora when you create a new object in the admin client and it contains its Dublin Core metadata.

Looking at the main part of the page there is little new here but note that the datastream is, of necessity, 'internal XML metadata'. If you click the 'edit' button at the bottom of the page the metadata itself is revealed:

```
<oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
                       xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
  <dc:title>Coliseum in Rome</dc:title>
  <dc:creator>Thornton Staples</dc:creator>
  <dc:subject>Architecture, Roman</dc:subject>
  <dc:description>Image of Coliseum in Rome</dc:description>
  <dc:publisher>University of Virginia Library</dc:publisher>
  <dc:format>image/jpeg</dc:format>
  <dc:identifier>demo:29</dc:identifier>
</oai_dc:dc>
```

As we shall see later, this object has more Dublin Core metadata than the admin client provides by default (which is only the title, and identifier). The rest has been added by some other means - perhaps, even, through this 'edit' tool.

Disseminators

The final tab on our Fedora object is labelled 'disseminators'. If an object in the repository is to be retrieved in anything but a passive way - for instance, typing its unique Fedora URL into a browser, then it needs to have one or more disseminators associated with it in order to define these non-passive behaviours.

If you open the 'disseminators' tab and have a look at 'DISS1' you will see that at least one 'clever' behaviour for our object is defined by another Fedora object 'demo:27'. In fact demo:27 defines several behaviours, one of which is called 'grayscaleImage'. You can see what this does by typing into your browser:

http://*yourBaseURL:port*/fedora/get/demo:29/demo:27/grayscaleImage/

(Be careful with the US spelling of 'gray') Put crudely, get 'demo:29' and apply to it the behaviour defined by 'demo:27/grayscaleImage'.

Well, that's a not untypical Fedora object. Of course now you want to play with some. If you've played with the demonstration objects provided with the Fedora download you will have seen a number of behaviours used there and will probably see all sorts of possibilities for using them in your institution. But we need to be able to walk before we can run so let's take a time-honoured approach and try tinkering with some of the demonstrators first. Behaviours are probably the hardest part, so let's set up a few simple objects and see if we can manipulate them using the demonstration behaviours and leave writing our own until we have a much better grip of the basics.

Just before we do, a word to the wise. You have already had occasion to type 'demo:29' and the like several times. What is set in your Fedora configuration file as your prefix? Originally we had 'hull-repomman' but I quickly tired of typing that and shortened it to just 'hull'. Perhaps, especially whilst you're at the experimentation stage, you might like to consider something quite short, too!

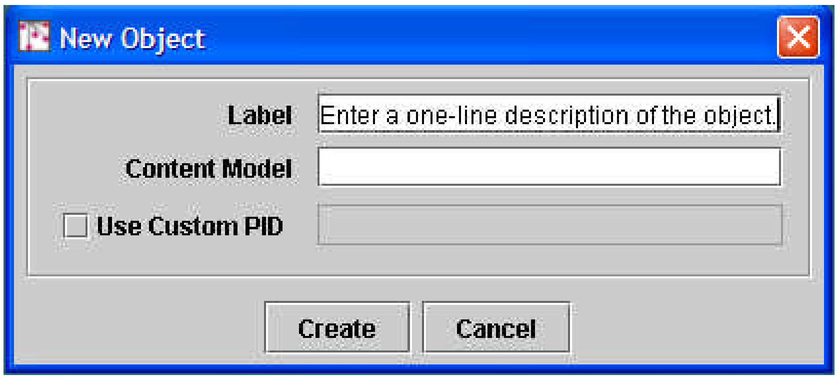
A first object (a managed object)

Many of the Fedora demonstration objects incorporate images and there are some behaviours that can be applied to them – so let us start there and create an object that – to begin with – incorporates a single jpeg image.

First of all find the jpeg image that you are going to use - preferably, for the purpose of this exercise with dimensions that will just fit nicely inside your normal browser window. I use a screen resolution of 1280x1024 and so I used 1000x750 for my image. The size is, in a sense, immaterial but it's probably better that the image will fit comfortably on your screen.

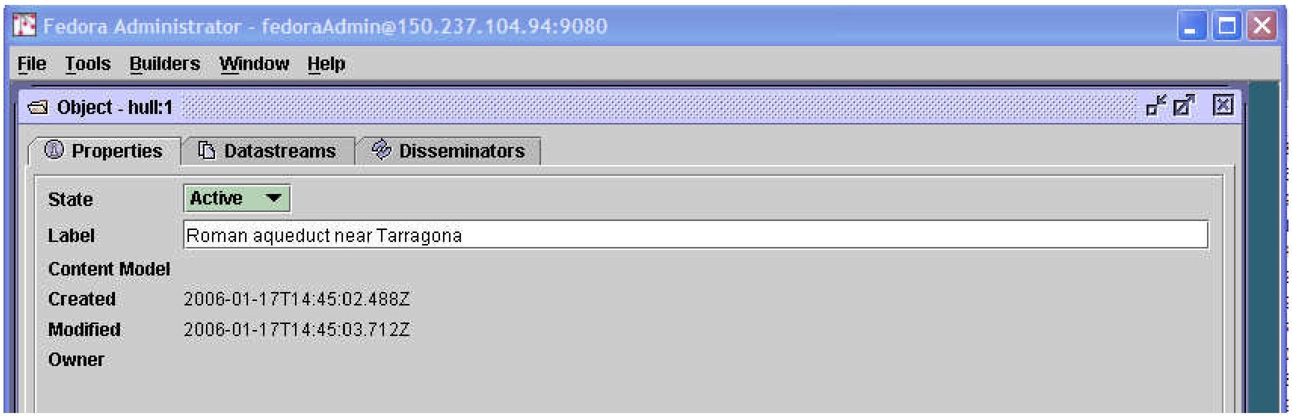
Create the basic object

In the Fedora admin client, select 'new' from the 'file' menu and then choose 'object'. The key combination 'ctrl-N' produces the same effect:



Label: Enter a one-line description of the object.
Content Model:
☐ Use Custom PID

I decided to follow the theme of 'demo:29' and entered into 'label' the text 'Roman aqueduct near Tarragona'. Obviously you will enter something appropriate to your image. This object will not need a content model, so that stays blank, and we'll worry about custom PIDs at a later stage - so if you're happy with what you've done, click 'create'. You should now have something like this...

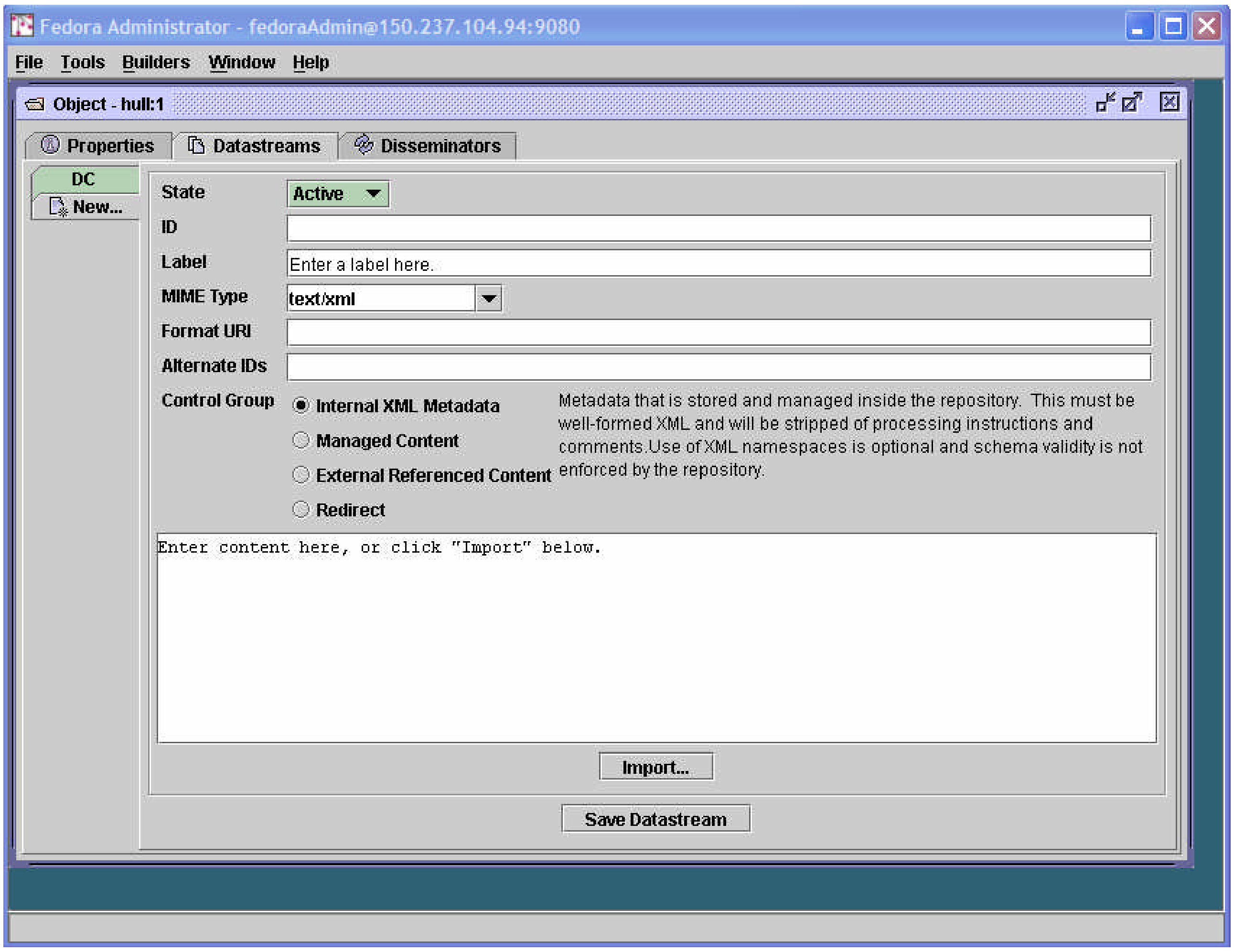

Fedora Administrator - fedoraAdmin@150.237.104.94:9080
File Tools Builders Window Help
Object - hull:1
Properties
Datastreams
Disseminators

State	Active
Label	Roman aqueduct near Tarragona
Content Model	
Created	2006-01-17T14:45:02.488Z
Modified	2006-01-17T14:45:03.712Z
Owner	

If you look in the 'datastreams' tab and look at the 'DC' datastream (you'll have to click 'edit' to see the content) you now have an object with very basic metadata - an entry for Dublin Core title and for identifier. Note on the 'DC' tab that the metadata has acquired a URL through which it can be accessed. Try typing it into a browser.

Creating a datastream within an object

Now we need to add our image, On the 'datastreams' page, click the 'new' tab:


Fedora Administrator - fedoraAdmin@150.237.104.94:9080
File Tools Builders Window Help
Object - hull:1
Properties
Datastreams
Disseminators
DC
New...
Import...
Save Datastream

State: Active
ID:
Label: Enter a label here.
MIME Type: text/xml
Format URI:
Alternate IDs:
Control Group: (●) Internal XML Metadata ( ) Managed Content ( ) External Referenced Content ( ) Redirect

Metadata that is stored and managed inside the repository. This must be well-formed XML and will be stripped of processing instructions and comments.Use of XML namespaces is optional and schema validity is not enforced by the repository.

```
Enter content here, or click "Import" below.
```

The first thing we have to enter is an 'ID'. This will become the label on the datastream tag - when you save the datastream it will replace the word 'New'. At this stage it is not critical what you enter here, but if you are going to be working with image objects it might be useful to develop a standard set of tags. Perhaps record on the tag the dimensions and file format 'jpg1000x750' (Be warned that the system does not seem to like datastream names that begin with a digit.)

The datastream itself now needs some sort of informative label.

You now need to set the MIME type for the data object. Unfortunately, if you 'pull down' the MIME type box you will find that you have no choice other than text/xml which is clearly inappropriate. In fact, you need to jump ahead to the 'control group' settings. The default is 'internal XML metadata' - hence the MIME type - but this is not what you want. Your image file will be managed by the repository and so you need to select 'managed content'. Once you have done that the large text box at the bottom of the screen will vanish and when you pull down the 'MIME type' list you will find a much longer list than before. Choose 'image/jpeg'.

Now you need to import the image itself. Click the 'import' button and point the admin client at your image. Once this has been done successfully your image (or part of it) will be visible at the bottom of the new datastream pane. Now you should save the datastream.

If all has gone according to plan you should now have a working digital object within your Fedora repository that can be retrieved by entering it's Fedora URL (from the bottom of the datastream record) into a browser or via the Fedora web search tool:

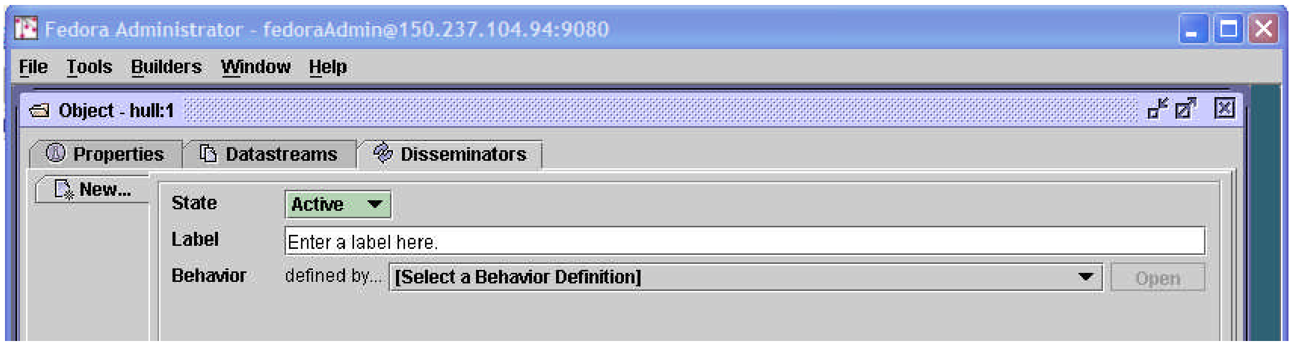
http://*yourBaseURL:port*/fedora/search

As it stands, this Fedora object is merely passive. It exists and it can be retrieved. For some people this may be enough, but if you were setting up an image repository then you might want to be able to manipulate your images in a number of ways. The next section will help you understand something of how you might do that.

Disseminating your object

To associate 'clever' behaviours with your repository object it is necessary to define at least one disseminator. The Fedora demonstration objects include a very clever set of behaviours in an object that we have already come across, demo:27 (remember 'grayscaleImage'?). Yes, behaviour definitions (bDefs) are themselves Fedora objects as are the behaviour mechanisms (bMechs) that they call upon. We are going to use these behaviours rather than attempt to write our own at this stage.

Switch to the 'disseminator' record of your object. At the moment there is no disseminator defined and you find yourself on the 'New' tab:


Fedora Administrator - fedoraAdmin@150.237.104.94:9080
File Tools Builders Window Help
Object - hull:1
Properties
Datastreams
Disseminators
New...
Open

State: Active
Label: Enter a label here.
Behavior: defined by... [Select a Behavior Definition]

Enter a label for the disseminator. (In my version of the software the system overwrites this, but this may not always be so.)

Pull down the list of behaviours - you will find that there is quite a variety. The one you want for now is 'demo:27 - Behaviour Definition Object for Image Manipulation Servlet'. Once you choose this a further part of the dissemination record is filled in. If you drop down the list for 'defines method' you will see that there is quite a range of methods defined in addition to just 'grayscaleImage'. You now need to select a behaviour mechanism and you will find that there is only one available for your purpose here. Once you have chosen it the last section of the record is displayed. It tells you that this behaviour mechanism requires binding to an image datastream. Well, you have one so click on 'add' and oops! Fedora says that you haven't. Didn't you just save your image datastream? Well, yes, you probably did. You have just discovered one of the admin client's little quirks. Objects with new datastreams must be saved before the datastream can be used in a disseminator. Call it a bug, call it a feature - it is what you have to do. So check that you know the name of the object (on the top bar of its window) and then close the object.

Now open it again (with 'open object' from the file menu or ctrl-O). Go to the disseminator record and, infuriatingly, your label and your previous clicks have been lost - but it was only two clicks... You may think making you lose this record was cruel of me - but now you will remember that new datastreams require the object to be saved. If had just told you that you probably wouldn't have remembered!

So, fill the record in again. This time when you try to add an image datastream you find that it is there to click on. Now you need to save the disseminator. Just before we leave it, let's have a look again at the list of methods available through demo:27:

- resizeImage
- zoomImage
- brightImage

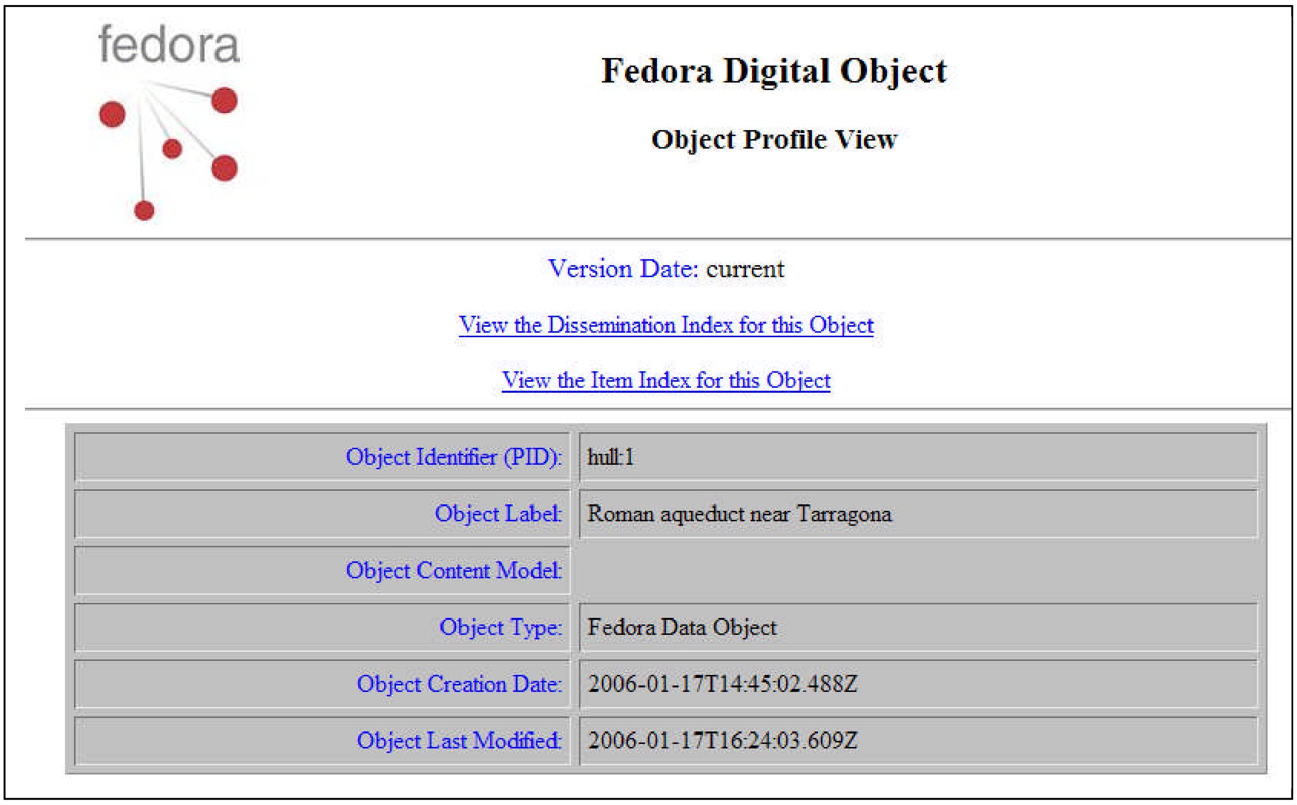
- watermarkImage
- grayscaleImage
- cropImage
- convertImage

Accessing your object

Let's now have a look at our object from the web. Open a browser and search your repository for it:

http://*yourBaseURL:port*/fedora/search

When you click its link you should get a page like this:



Fedora Digital Object

Object Profile View

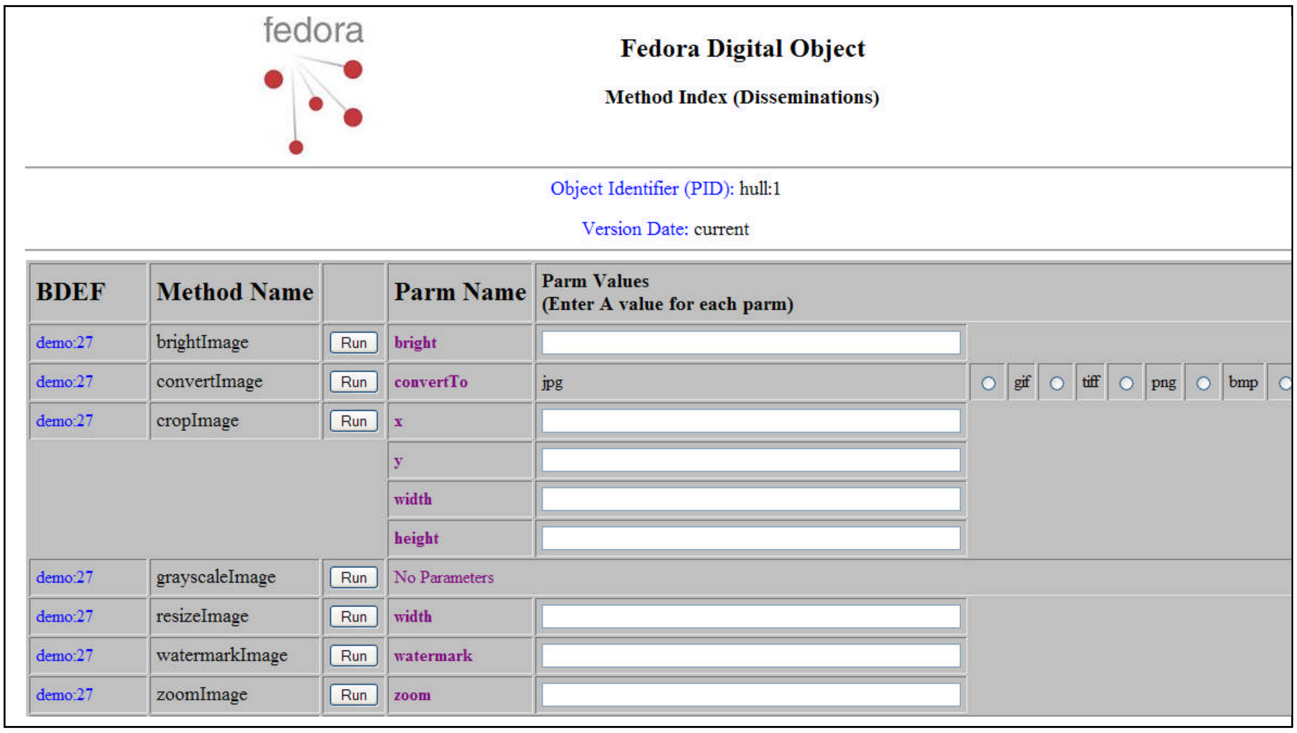
Version Date: current

View the Dissemination Index for this Object

View the Item Index for this Object

Object Identifier (PID):	hull:1
Object Label:	Roman aqueduct near Tarragona
Object Content Model:	
Object Type:	Fedora Data Object
Object Creation Date:	2006-01-17T14:45:02.488Z
Object Last Modified:	2006-01-17T16:24:03.609Z

If you view the Item Index you will find there your two datastreams (the Dublin core metadata and the image) whilst the Dissemination Index provides the basis for some further experimentation!



Fedora Digital Object

Method Index (Disseminations)

Object Identifier (PID): hull:1

Version Date: current

BDEF	Method Name		Parm Name	Parm Values (Enter A value for each parm)
demo:27	brightImage	Run	bright	
demo:27	convertImage	Run	convertTo	jpg ○ gif ○ tiff ○ png ○ bmp ○
demo:27	cropImage	Run	x	
			y	
			width	
			height	
demo:27	grayscaleImage	Run	No Parameters	
demo:27	resizeImage	Run	width	
demo:27	watermarkImage	Run	watermark	
demo:27	zoomImage	Run	zoom	

Before you can run 'brightImage' you need to supply a value in the box provided. A value of '1' gives you the unaltered image, '0.5' darkens it, '1.5' lightens it - and so on.

The entry for 'convertImage' should be self explanatory.

'cropImage' requires four parameters: the x and y values for the top left-hand corner of the crop (measured from the top left-hand corner of the original image) and the width and height of the crop - all in pixels.

'grayscaleImage' is self-explanatory.

'resizeImage' requires a width, in pixels, for the resized image. the proportions of the original are retained.

'watermarkImage' requires the text for a watermark.

'zoomImage' requires a scaling factor. '1' is full size, '0.5' is half size, '2' is twice size - but note that the zoom occurs within the limits of the original image size.

So, adding a dissemination method to an object opens up all sorts of possibilities!

Another disseminator

Let's create a few more objects containing digital images and look at adding a different disseminator to them. In setting up the objects we will be preparing the way to look at the idea of collections - so you probably should identify some sort of theme that you want to pursue.

Create a group of objects

First of all, you will need a number of objects (three or four), with a similar theme, that you can then later 'collect' together (in the Fedora sense). One of my interests is garden roses, so I started with a couple of rose photos. For each digital object you will need a reasonably high resolution image and a smaller thumbnail version of it (say 300x225 pixels).

First of all, using the techniques that we described in the first section, make a simple digital object for each pair of photos.

For each one:

- create the basic object using the 'new' dialogue box. Each will need an appropriate label but it should not have a content mode
- on the 'datastreams' record create a datastream for the thumbnail version of the image. The datastream needs a sensible name (something like jpg300x225 if you follow the suggestion made above), it needs to be in the control group 'managed content', and it needs to have an 'image/jpeg' MIME type

When you have got this far you might want to check that the object 'works' by typing its Fedora URL

http://*yourBaseURL:port*/fedora/get/*yourObject:nn/jpg300x225*

into a browser. Assuming all is OK, go back to the object in the admin client and add a second datastream just like the first but with a different and appropriate name (say jpg1000x750).

Again, you might want to check that the second datastream works:

http://*yourBaseURL:port*/fedora/get/*yourObject:nn/jpg1000x750*

Another interesting check would be to use the Fedora search interface:

http://*yourBaseURL:port*/fedora/search

to search for the new object. Have a look at its 'item index' and check that, by clicking the links for the two images datastreams, you can display both images on screen.

If that has worked, then create one or two more themed objects in the same way.

Add some extra metadata

The next stage in this process is to make sure that all your objects have the necessary Dublin Core metadata for the dissemination method that we shall eventually use with our first collection. As you created them Fedora will have given your objects a title, based on the first label that you provided, and will have assigned an identifier - Fedora's unique name for the object.

If I open up (click the 'edit' button) the metadata for the Roman aqueduct object that I described earlier it looks like this:

```
<oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
                         xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
  <dc:title>Roman aqueduct near Tarragona</dc:title>
  <dc:identifier>hull:1</dc:identifier>
</oai_dc:dc>
```

You can see the title enclosed in tags <dc:title></dc:title> and the identifier enclosed by <dc:identifier></dc:identifier>tags. You need to add in a description enclosed by <dc:description></dc:description>tags. Thus mine might become:

```
<oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
                         xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
  <dc:title>Roman aqueduct near Tarragona</dc:title>
  <dc:identifier>hull:1</dc:identifier>
  <dc:description>This is a photograph of the 'Pont du Diable' (Devil's Bridge), a
                         Roman aqueduct just outside Tarragona in Spain.</dc:description>
</oai_dc:dc>
```

You may realise, from your own look around Fedora, that there are a number more Dublin Core items that might usefully belong here. We shall come back to this subject; all we have done here is to make sure that we have the three items that you will need when you disseminate your collection later. This extra item of descriptive metadata needs to be added to each of your objects.

Add the disseminator

As with the first object that we created, we are going to use one of the disseminators that comes in amongst the demonstration objects that Fedora provides. This one is designed to deal with digital objects that contain a high resolution image and a thumbnail without the end user needing to know what the datastreams within the object are called; different objects may well have them labelled differently (even if we might consider that dubious practice).

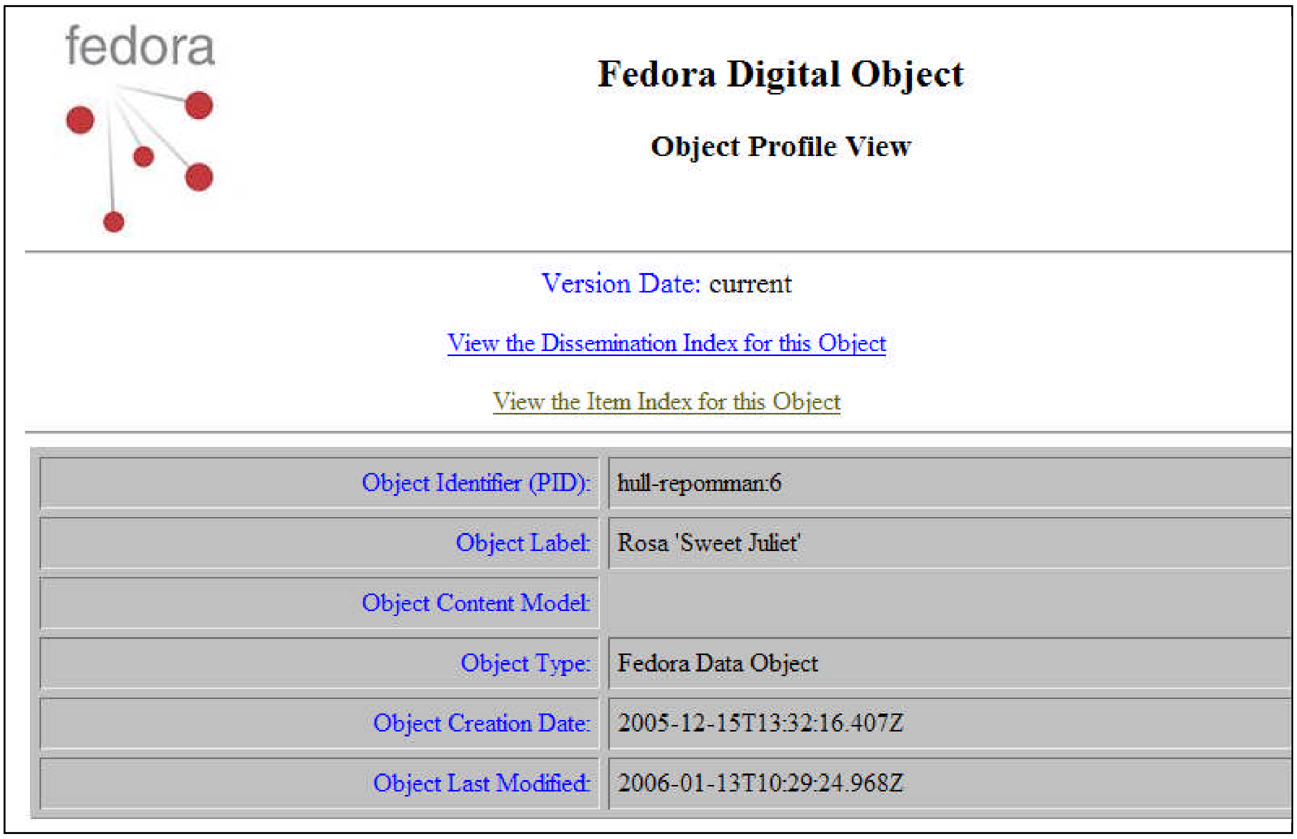
You will need to repeat this process for each of your objects and (you may remember) it is essential that your object has been closed (and therefore saved) since the image datastreams were set up.

- Go to the 'Disseminators' record and add a new disseminator which must have a label of 'DualResImage'.
- From the 'behaviours' list select 'demo:DualResImage - Dual Resolution Image.
- From the 'mechanisms' list select 'demo:DualResImageImpl - Dual Resolution Image Implementation' which is probably the only behaviour listed.
- The screen will then ask you to add bindings for 'FULL_SIZE', so add the larger of your image datastreams (jpg1000x750?), then add a binding to MEDIUM_SIZE for the smaller image (jpg300x225?).
- Save the disseminator.

Remember that if your image datastreams are not listed when you get to the third bullet point, you probably haven't saved your digital object after creating the image datastreams.

Let's test it out

Open up the Fedora web search tool and search your repository to find one of the new objects. Open it up. You should get a screen like this:



Fedora Digital Object

Object Profile View

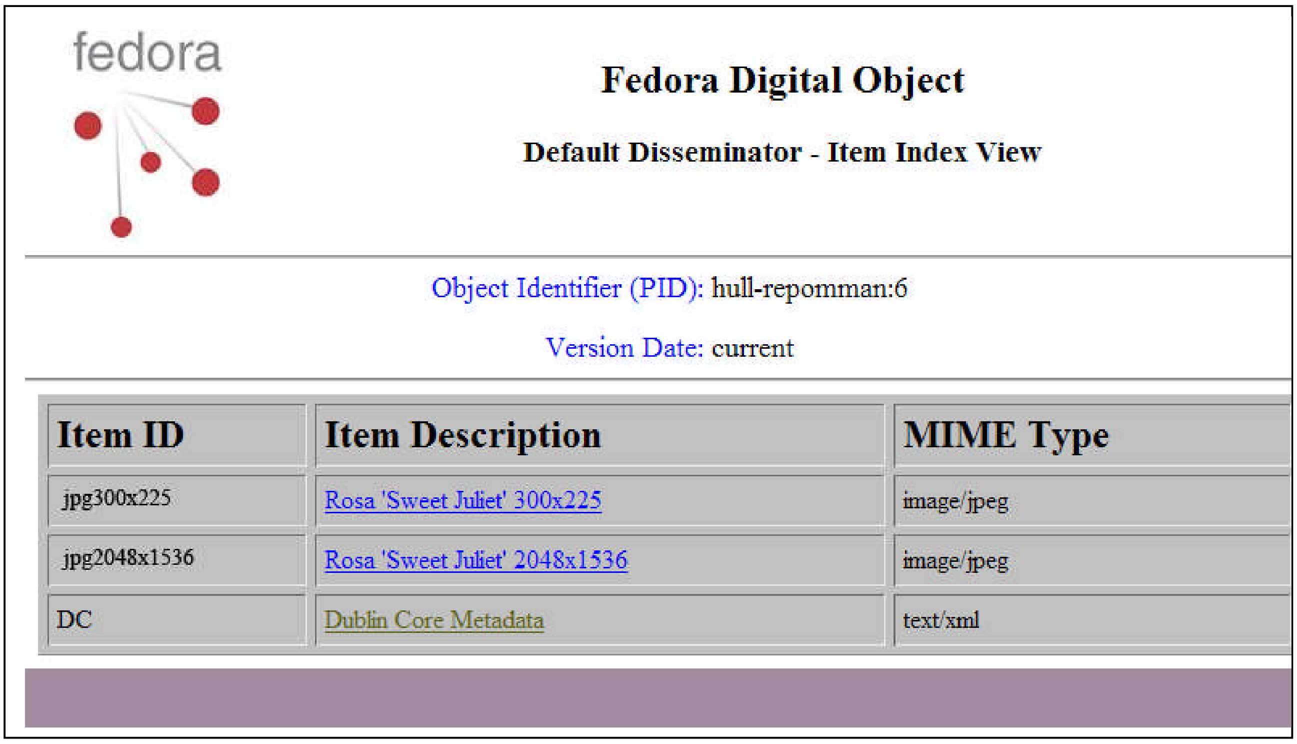
Version Date: current

View the Dissemination Index for this Object

View the Item Index for this Object

Object Identifier (PID):	hull-repomman:6
Object Label:	Rosa 'Sweet Juliet'
Object Content Model:	
Object Type:	Fedora Data Object
Object Creation Date:	2005-12-15T13:32:16.407Z
Object Last Modified:	2006-01-13T10:29:24.968Z

First of all, have a look at the item index for the object - it will look something like this:



Fedora Digital Object

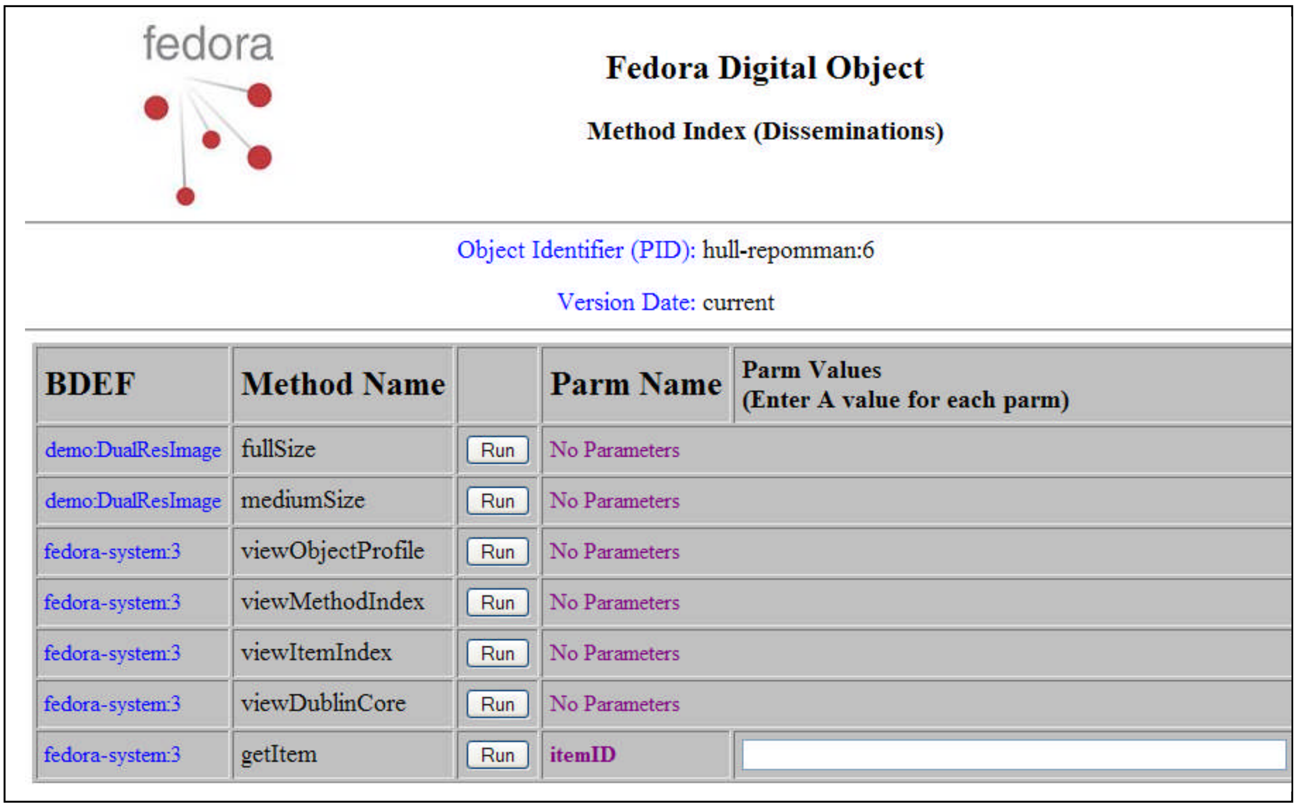
Default Disseminator - Item Index View

Object Identifier (PID): hull-repomman:6

Version Date: current

Item ID	Item Description	MIME Type
jpg300x225	Rosa 'Sweet Juliet' 300x225	image/jpeg
jpg2048x1536	Rosa 'Sweet Juliet' 2048x1536	image/jpeg
DC	Dublin Core Metadata	text/xml

If you click each of the 'item description' links they will open that datastream and display it for you. So why, you ask, do we need a disseminator? If you look closely at the diagram above, you will see that my large image is not called 'jpg1000x750' as I suggested that yours might be. Indeed, you may not have used resolution of 1000 x 750 for your large image and therefore likewise have a different name. How is a user supposed to know what you have called the datastreams in a particular object? Answer, there is no need. Go back one page in your browser and click to look at the 'dissemination index for this object'.



Fedora Digital Object

Method Index (Disseminations)

Object Identifier (PID): hull-repomman:6

Version Date: current

BDEF	Method Name		Parm Name	Parm Values (Enter A value for each parm)
demo:DualResImage	fullSize	Run	No Parameters	
demo:DualResImage	mediumSize	Run	No Parameters	
fedora-system:3	viewObjectProfile	Run	No Parameters	
fedora-system:3	viewMethodIndex	Run	No Parameters	
fedora-system:3	viewItemIndex	Run	No Parameters	
fedora-system:3	viewDublinCore	Run	No Parameters	
fedora-system:3	getItem	Run	itemID	

Here we find two 'methods' associated with the disseminator that we used for our objects: 'fullSize' and 'mediumSize'. If you run them you will again see the contents of the datastreams - but we did not need to know what they were called, you made that binding inside the object so that the user would not need to know anything about it.

Collections

One of the oft-quoted strengths of Fedora is the way it can manage 'collections' of digital objects, that is groups of objects that are related in some way. At the time of writing this area of Fedora's functionality was not well documented by the development team.

Again it seems to make sense to follow the path of adapting some of the Fedora demonstration material as a way to understanding how collections might work for us. You will need to be reasonably comfortable with the ideas developed in the previous sections about creating a digital object and adding a disseminator to it - but, of course, you can always refer back if you need to!

Create a collection object

Well, thus far we have two or three objects with a loose theme, each having the demonstration Dual Resolution disseminator embedded. How do we make Fedora recognise them as a collection?

Stage one is to define another digital object within Fedora - a collection object. Think of it like this: if a user correctly searches the repository for a particular picture that is in there, it will be found. To search for a collection, there must similarly be an object representing that collection to find. How the collection object relates to its component parts is something that we shall deal with in a little while. First, let's create it.

Create a new object in the usual way with an appropriate label - I called mine 'Rose Collection - explicit'. Because of the way we are going to disseminate this collection object we again need to add a Dublin Core description line into the metadata. Mine was:

```
<dc:description>This is an explicit collection (of roses)</dc:description>
```

And that's it, in one sense. There is a collection object that could be found by a search. (Indeed if you use the Fedora search tool you will find it there.) However it doesn't yet do anything, it just 'is', it exists. Furthermore, it knows nothing about the group of digital objects that it is supposed to represent. We will address that problem now.

Explicit and implicit collections

There are two ways in which a basic collection can be expressed and which you use will probably depend on circumstances and local needs.

An explicit collection is defined by listing its member objects within the collection object. This is probably the most obvious way to make a collection but it is not terribly flexible. If you were to add a new object to the collection you would need not only to create it, but also to alter the collection object itself - so you have two objects that you need to work on. Implicit collections (watch this space) are a better way of dealing with collections whose membership may change with time. Explicit collections are best used for collections which have a fixed and unchanging membership. So, for instance, you might want to use an explicit collection for 'The Works of Shakespeare'. This collection would have a finite, unchanging number of members. (OK, so there is an outside chance that another one might be found, but...) So maybe my use of roses for an explicit collection was not an entirely appropriate one, but we needed something to collect as an example

An implicit collection works the other way up. Instead of the collection object listing the members, the members list the collection object. In other words, each member of the collection individually claims "I am a member of the xxx collection". This is good for a collection like my cats photos. (For many years I have shared my home with two or three cats at any given time. We shall come to them soon!) I never know when I might want to add another member, but when I do I need only to create the new object and have it assert that it is a member of my cats collection. I don't need to manipulate the collection object itself. Now this may have you scratching your head and asking "So how, if my search finds the collection object, would it know about its members?" For the moment, all I can say is "Trust me!" The answer is a little too complex to get diverted by it here and, in any case, you will perhaps come to understand it better if we have first dealt with explicit objects fairly thoroughly.

Creating the relationships for an explicit collection

How do we go about associating the members of an explicit collection with the collection object itself. For this, the collection object needs a special datastream called 'RELS-EXT' (relationships - external).

Open up your collection object and go to the datastreams record. Create a new datastream with an id 'RELS-EXT'. This is a particular form of datastream that Fedora understands and no other name will do. I tend to label mine 'External relationships'. The control group needs to be 'internal XML metadata' and the MIME type 'text/XML'. You need to type in the metadata to look like this:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rel="info:fedora/fedora-system:def/relations-external#">
  <rdf:Description rdf:about="info:fedora/hull-repomman:20">
    <rel:hasMember rdf:resource="info:fedora/hull-repomman:6"/>
    <rel:hasMember rdf:resource="info:fedora/hull-repomman:8"/>
  </rdf:Description>
</rdf:RDF>
```

Let's look carefully at this chunk of XML.

The whole is wrapped in a pair of tags which define the 'name spaces' for the XML.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rel="info:fedora/fedora-system:def/relations-
external#">
</rdf:RDF>
```

Now is not the time to go into name spaces. If you are an XML buff you will know all about them, if not just accept for the moment that this outside pair of tags need to be there 'as is', character-for-character perfect.

Within these tags are a pair of 'Description' tags which define which object these external relationships apply to.

```
<rdf:Description rdf:about="info:fedora/hull-repomman:20">
</rdf:Description>
```

Your opening description tag needs to contain the name of the collection object that you are working with (in my case 'hull-repomman:20') - you can find it on the top bar of the object's open window in your admin client, just repeat it in the metadata tag.

Between the 'Description' tags you need to list the members of the collection using multiple tags of the form:

```
<rel:hasMember rdf:resource="info:fedora/hull-repomman:6"/>
```

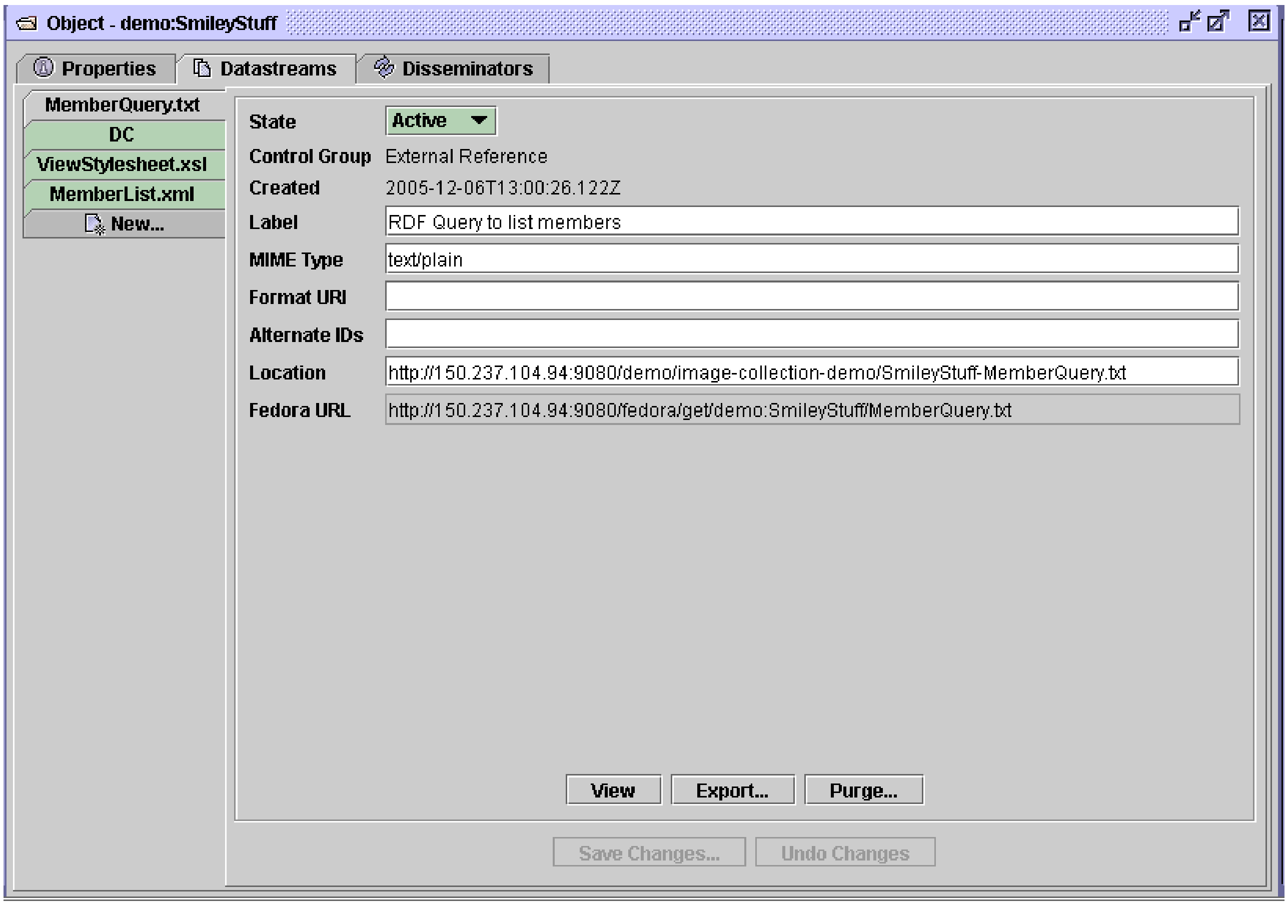
...except, of course, that you will need to substitute the names of your own objects where my example says 'hull-repomman:6'. These lines assert that the collection object (in my case, 'hull-repomman:20') has a finite list of members ('hull-repomman:6' and 'hull-repomman:8') - just two in this case for the sake of simplicity. You will probably want to add all two or three that I suggested you create.

So there we have it: a collection object that knows about a list of members, and the member objects themselves. But these are just passive objects - they don't do anything much other than exist. To see one way in which collections might be used, we are going to add a dissemination behaviour.

A collection disseminator

What we are going to do now is to add the demonstration collection disseminator to our collection object. It was round about here in the process that I started feeling I needed to go into a corner periodically with a cold compress on my head - but I'll try to spare you some of the pain.

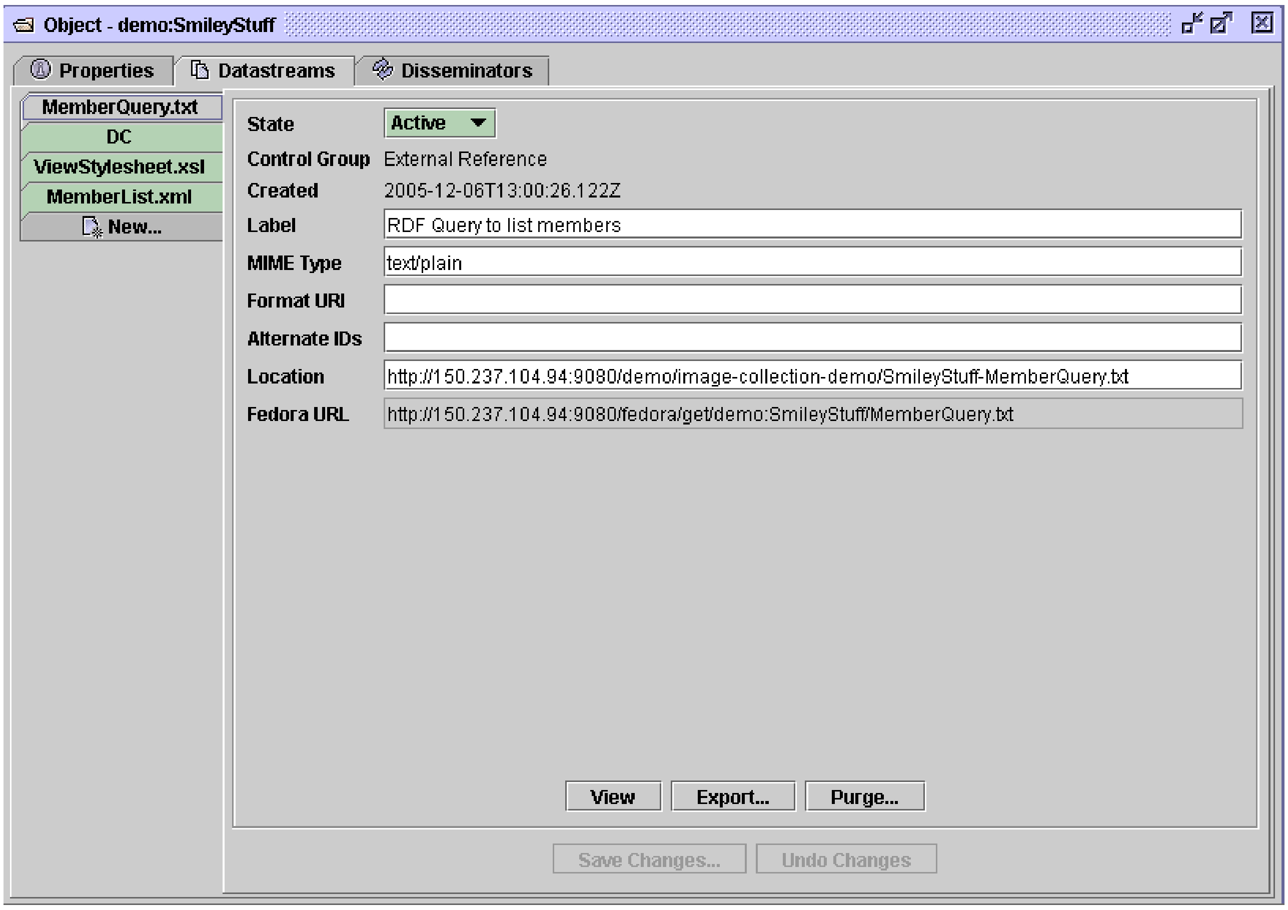
As before, we are going to take the Fedora demonstration material and adapt it stage by stage. The first thing we need to do is to look at the collection object provided by Fedora for their collection of dual resolution 'Smiley' photographs. It is called 'demo:SmileyStuff'. If you open it up in the admin client and have a look at its 'datastreams' record, you should find this:



State: Active
Control Group: External Reference
Created: 2005-12-06T13:00:26.122Z
Label: RDF Query to list members
MIME Type: text/plain
Format URI:
Alternate IDs:
Location: http://150.237.104.94:9080/demo/image-collection-demo/SmileyStuff-MemberQuery.txt
Fedora URL: http://150.237.104.94:9080/fedora/get/demo:SmileyStuff/MemberQuery.txt

Apart from the normal Dublin Core datastream, three additional datastreams exist. (The Smiley materials are actually an example of an implicit collection and so this collection object does not need the 'RELS-EXT' datastream that you have in yours to define the collection members. That apart, it is essentially the same.)

If you have a look at the 'disseminators' record you will see that we are going to use a behaviour definition called 'demo:Collection' and a behaviour mechanism specifically for a dual resolution image collection. This needs three bindings corresponding to the three extra datastreams. Let's go back to them and have a look first at 'memberQuery.txt'.



State: Active
Control Group: External Reference
Created: 2005-12-06T13:00:26.122Z
Label: RDF Query to list members
MIME Type: text/plain
Format URI:
Alternate IDs:
Location: http://150.237.104.94:9080/demo/image-collection-demo/SmileyStuff-MemberQuery.txt
Fedora URL: http://150.237.104.94:9080/fedora/get/demo:SmileyStuff/MemberQuery.txt

This is described as an RDF query to list members. If you open it with the 'view' button, you will find the RDF query itself - but before we move on, note two things. This is in the control group 'external reference' and it has a 'location' - not something we have seen before. The location here is actually part of the Fedora website structure.

Let's have a look at the query itself:

```
select $collTitle $collDesc $member $memberTitle $memberDesc
from   <#ri>
where  <info:fedora/demo:SmileyStuff>   <dc:title>                  $collTitle
and     <info:fedora/demo:SmileyStuff>  <dc:description>            $collDesc
and    $member                          <fedora-rels-ext:isMemberOf>
                                                  <info:fedora/demo:SmileyStuff>
and    $member                          <dc:title>                  $memberTitle
and    $member                          <dc:description>            $memberDesc
```

If you don't know much about RDF and triple stores (I didn't, but I'm learning fast), this is where it all begins to look a bit like black magic. But let's persevere. Remember that this is for an implicit collection, which we haven't dealt with yet. We're dealing with the slightly easier case of an explicit collection and it turns out that you need something similar, but slightly different. The RDF query I needed was this:

```
select $collTitle $collDesc $member $memberTitle $memberDesc
from    <#ri>
where   <info:fedora/hull-repomman:20>   <dc:title>                     $collTitle
and     <info:fedora/hull-repomman:20>   <dc:description>               $collDesc
and     <info:fedora/hull-repomman:20>   <fedora-rels-ext:hasMember>    $member
and     $member                          <dc:title>                     $memberTitle
and     $member                          <dc:description>               $memberDesc
```

As I understand it, in English it goes something like this.

```
select $collTitle, $collDesc etc (a set of variables which we can define)
from       #ri (Fedora's resource index)
objects
where      <info:fedora/hull-repomman:20>   has a   <dc:title>               put it in   $collTitle
and        <info:fedora/hull-repomman:20>   has a   <dc:description>         put it in   $collDesc
and        <info:fedora/hull-repomman:20>   has a   <fedora-rels-ext:hasMember> relationship -
                                                       store the member pid in           $member
and this   $member                          has a   <dc:title>               put it in   $membertitle
and this   $member                          has a   <dc:description>         put it in   $memberDesc
```

When one matching record has been found, the system looks for another until it runs out.

None of this query is optional; all the component parts must be found. Remember that we had to put in some <dc:description> metadata? As this query is written, items without a <dc:description> would not be found.

The variables for each object found are written out to an XML file which looks like this:

```
<sparql xmlns="http://www.w3.org/2001/sw/DataAccess/rf1/result">
  <head>
    <variable name="collTitle"/>
    <variable name="collDesc"/>
    <variable name="member"/>
    <variable name="memberTitle"/>
    <variable name="memberDesc"/>
  </head>
  <results>
    <result>
      <collTitle>Rose collection - explicit.</collTitle>
      <collDesc>This is an explicit collection (of roses)</collDesc>
      <member uri="info:fedora/hull-repomman:6"/>
      <memberTitle>Rosa 'Sweet Juliet'</memberTitle>
      <memberDesc>This is my favourite old rose - rosa 'Sweet Juliet'</memberDesc>
    </result>
    <result>
      <collTitle>Rose collection - explicit.</collTitle>
      <collDesc>This is an explicit collection (of roses)</collDesc>
      <member uri="info:fedora/hull-repomman:8"/>
      <memberTitle>Rosa 'Isobel Derby'</memberTitle>
      <memberDesc>This is the rose 'Isobel Derby'</memberDesc>
    </result>
  </results>
</sparql>
```

Let's recap. The RDF query interrogates the Fedora resource index and returns an XML file with the results. The RDF query itself is an external file located on the Fedora server.

Your first job is to write an adapted file (in WordPad or similar) and save it as a simple text file called Roses-MemberQuery.txt or similar. (You aren't collecting roses, probably.) All you should need to alter is the name of the collection object (which occurs three times, it is info:fedora/hull-repomman:20 in my file). Where to put it? On our server configuration it is in the directory:

/usr/local/fedora-2.0/server/jakarta-tomcat-5.0.28/webapps/ROOT/repomman

You need to store your query file in the equivalent place. Now create the datastream in your collection object:

- create a new datastream with an ID of 'MemberQuery.txt' (the name needn't be identical because you are going to create a binding, but why not?).
- make the control group 'external referenced content'
- give it a label of your choice
- set the MIME type to 'text/plain', and
- set the location to
  http://*yourBaseURL:port*/repomman/Roses-MemberQuery.txt

For peace of mind, click the 'view' button at the bottom of the datastream record and check that it finds your file correctly.

Now you need to tell your collection object where the result of this query, the XML, can be found. Create another datastream and call it 'MemberList.xml'. This is very similar to the last one except that the MIME type will be 'text/xml' and the location is:

http://*yourBaseURL:port*/fedora/get/*yourObject:nn*/demo:Collection/list

In other words it can be found by running the 'list' method of 'demo:Collection' against your collection object.

Finally - we need a datastream called 'ViewStylesheet.xsl', control group 'external referenced content', MIME type 'text/xml' which references the file at:

http://*yourBaseURL:port*/fedora/get/*yourObject:nn*/ViewStylesheet.xsl

But, you cry, "we don't have such a file!" Happily, Fedora have provided this one for us at:

/usr/local/fedora-2.0/server/jakarta-tomcat-5.0.28/webapps/ROOT/demo/ (continued)
image-collection-demo/SmileyStuff-ViewStylesheet.xsl

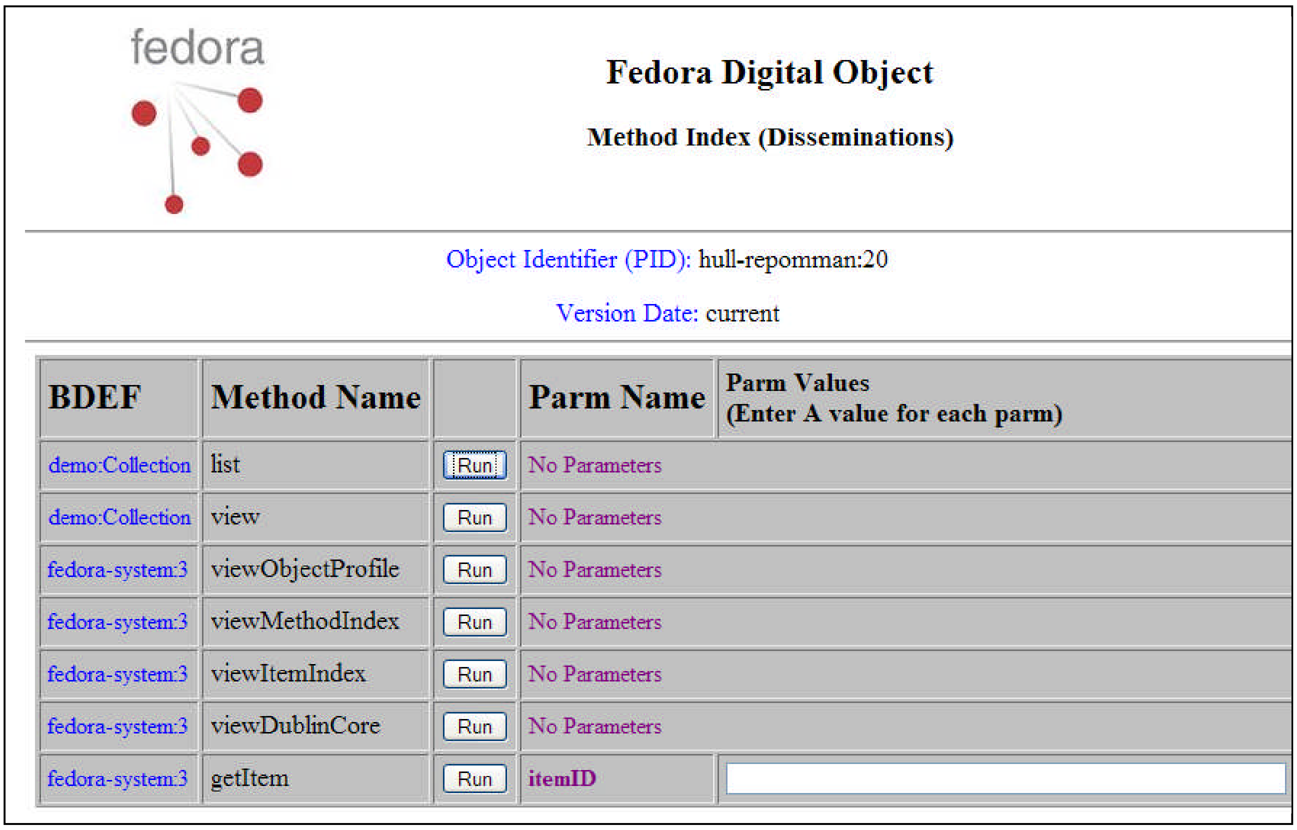
Copy it to the directory that contains your query and rename it simply 'ViewStylesheet.xsl'. This xsl file takes the XML generated as a result of the query into your list and displays it as an HTML page for the user.

Right, now we need to set up the disseminators - but before we try that, close your collection object and re-open it. Remember that the Fedora admin client requires you to do this before you can see the new datastreams for binding.

- create a new disseminator with an id of DISS1
- whose behaviour is defined by 'demo:Collection (Collection of Objects)'
- using the mechanism 'demo:DualResImageCollection'
- bind 'QUERY' to your 'MemberQuery.txt'
- bind 'LIST' to your 'MemberList.xml', and
- bind 'XSLT' to your 'ViewStylesheet.xsl'

Hopefully, all is now done. Our collection has a dissemination method associated with it and the datastreams are there for the dissemination to use. To see the effect, you need to search for your collection object using a browser http://*yourBaseURL:port*/fedora/search

and have a look at its dissemination methods. My roses collection object comes up as:



fedora

Fedora Digital Object

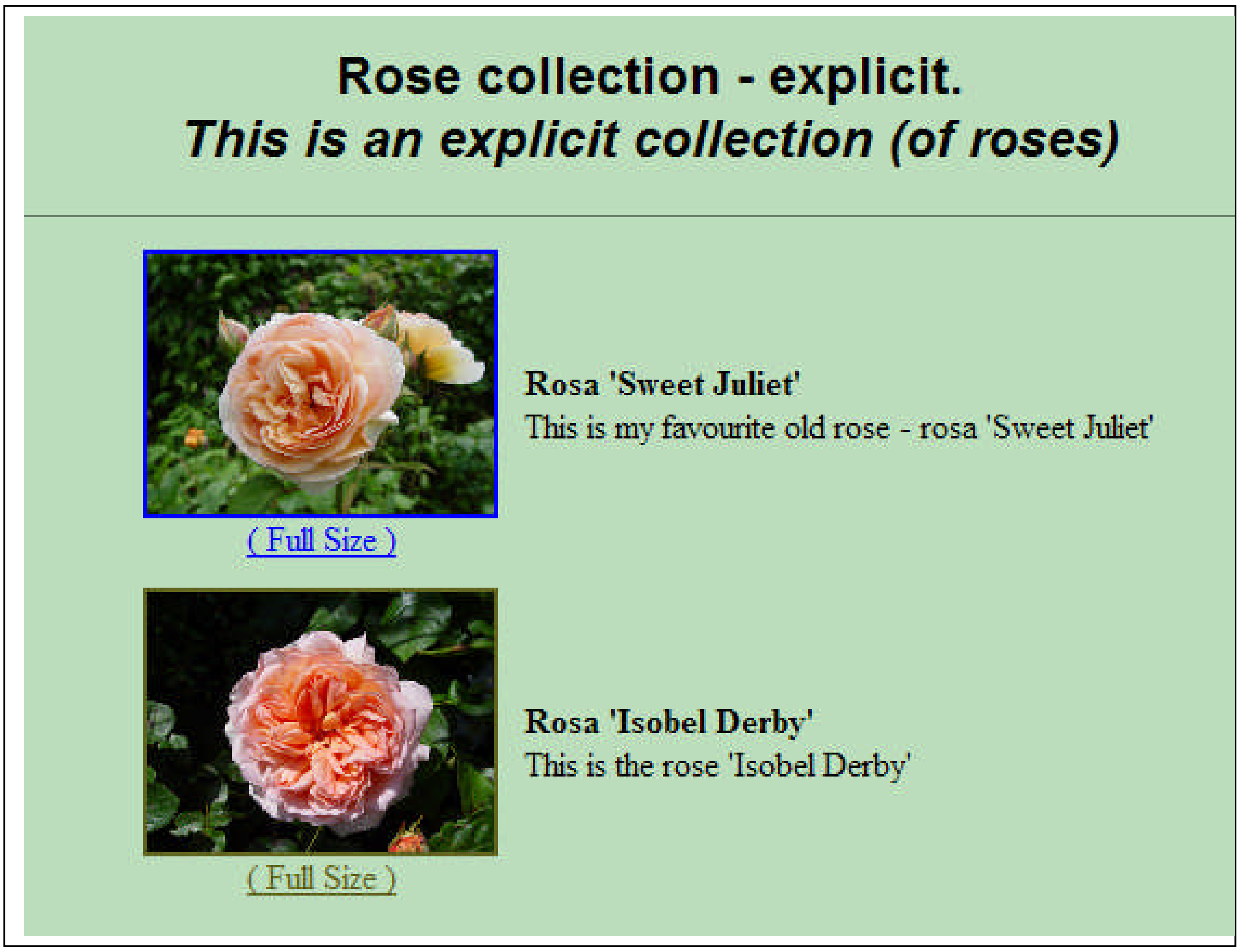
Method Index (Disseminations)

Object Identifier (PID): hull-repomman:20

Version Date: current

BDEF	Method Name		Parm Name	Parm Values (Enter A value for each parm)
demo:Collection	list	Run	No Parameters	
demo:Collection	view	Run	No Parameters	
fedora-system:3	viewObjectProfile	Run	No Parameters	
fedora-system:3	viewMethodIndex	Run	No Parameters	
fedora-system:3	viewItemIndex	Run	No Parameters	
fedora-system:3	viewDublinCore	Run	No Parameters	
fedora-system:3	getItem	Run	itemID	

If you run the 'list' method, you should get the XML listing that we saw above. If you run the 'view' method you will see the result of applying the xsl stylesheet to the list:



Rose collection - explicit.
This is an explicit collection (of roses)

Rosa 'Sweet Juliet'
This is my favourite old rose - rosa 'Sweet Juliet'

(Full Size)

Rosa 'Isobel Derby'
This is the rose 'Isobel Derby'

(Full Size)

Now that was worth all the hard work, wasn't it? At last we can see how the power of Fedora might be harnessed to do something more than just passively retrieve an object or two! If you compare the contents of the XML file with the screenshot, you will see how the information has been fitted into the HTML page.

Creating the relationships for an implicit collection

Our example above shows what can be done with an explicit collection of objects, one where the collection object itself lists all the members of the grouping. Let us now turn our attention to an implicit collection, one where each member of the group asserts its relationship with the collection object. In an attempt to keep life fairly simple, let us parallel exactly the development of the explicit collection above.

First job is to create a small group of image objects, each with a high resolution and a thumbnail datastream. Add the missing <dc:description></dc:description> metadata for each and add the dual resolution dissemination information to each.

Now create a collection object for these new objects and add its <dc:description></dc:description>.

Now we need to set about making our collection. As I said some time ago, we share our home with two or three cats and so have an expanding collection of cat photos. I used four of these as the basis for my implicit collection.

The collection object has a pid 'hull-repomman:13'. It has no RELS-EXT datastream because it does not define its own members. Instead, you will need to add a RELS-EXT datastream to each of the objects in your collection. If I open up one of my cat objects, 'hull-repomman:9', the RELS-EXT datastream that I have added looks like this:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rel="info:fedora/fedora-system:def/relations-external#">
  <rdf:Description rdf:about="info:fedora/hull-repomman:9">
    <rel:isMemberOf rdf:resource="info:fedora/hull-repomman:13"/>
  </rdf:Description>
</rdf:RDF>
```

The relationship is 'about' the object itself ('hull-repomman:9') and it 'isMemberOf' 'hull-repomman:13', my cats collection object. So you need to add something equivalent to your themed objects.

Now you have to add dissemination methods to your collection object. We are going to follow exactly the routine that we followed for the explicit collection above, so the first thing we need is an RDF query (which will become the MemberQuery.txt datastream) to find the collection objects. Mine looks like this:

```
select $collTitle $collDesc $member $memberTitle $memberDesc
from    <#ri>
where   <info:fedora/hull-repomman:13>  <dc:title>              $collTitle
and     <info:fedora/hull-repomman:13>  <dc:description>        $collDesc
and     $member                         <fedora-rels-ext:isMemberOf>
                                                  <info:fedora/hull-repomman:13>
and     $member                         <dc:title>              $memberTitle
and     $member                         <dc:description>        $memberDesc
```

In English it goes something like this.

```
select $collTitle, $collDesc etc (a set of variables which we can define)
from      #ri (Fedora's resource index)
objects
where     <info:fedora/hull-repomman:13>  has a        <dc:title>          put it in  $collTitle
and       <info:fedora/hull-repomman:13>  has a        <dc:description>    put it in  $collDesc
and       a member (put it in $member)    which has a  <fedora-rels-ext:isMemberOf> relationship
                                                          with  <info:fedora/hull-repomman:13>
and this  $member                         has a        <dc:title>          put it in  $membertitle
and this  $member                         has a        <dc:description>    put it in  $memberDesc
```

When one matching record has been found, the system looks for another until it runs out.

So, write yourself an RDF query in Wordpad, or similar, save it as a plain text file and store it in the appropriate directory on your server using an appropriate name. On our server configuration you may remember it needs to be in the directory:

/usr/local/fedora-2.0/server/jakarta-tomcat-5.0.28/webapps/ROOT/repomman

You need to store your query file in the equivalent place.

Now create the datastream in your collection object:

- create a new datastream with an ID of 'MemberQuery.txt'
- make the control group 'external referenced content'
- give it a label of your choice
- set the MIME type to 'text/plain', and
- set the location to
  http://*yourBaseURL:port*/repomman/Cats-MemberQuery.txt

or whatever you called your query.

For peace of mind, click the 'view' button at the bottom of the datastream record and check that it finds your file correctly.

Now you need to tell your collection object where the result of this query, the XML, can be found. Create another datastream and call it 'MemberList.xml'. This is very similar to the last one except that the MIME type will be 'text/xml' and the location is:

http://*yourBaseURL:port*/fedora/get/*yourObject:nn*/demo:Collection/list

In other words it can be found by running the 'list' method of 'demo:Collection' against your collection object.

Finally - we need a datastream called 'ViewStylesheet.xsl', control group 'external referenced content', MIME type 'text/xml' which references the file at:

http://*yourBaseURL:port*/fedora/get/*yourObject:nn*/ViewStylesheet.xsl

If you have been following this guide steadily, you will already have copied this into the appropriate directory and there is no need to do it again.

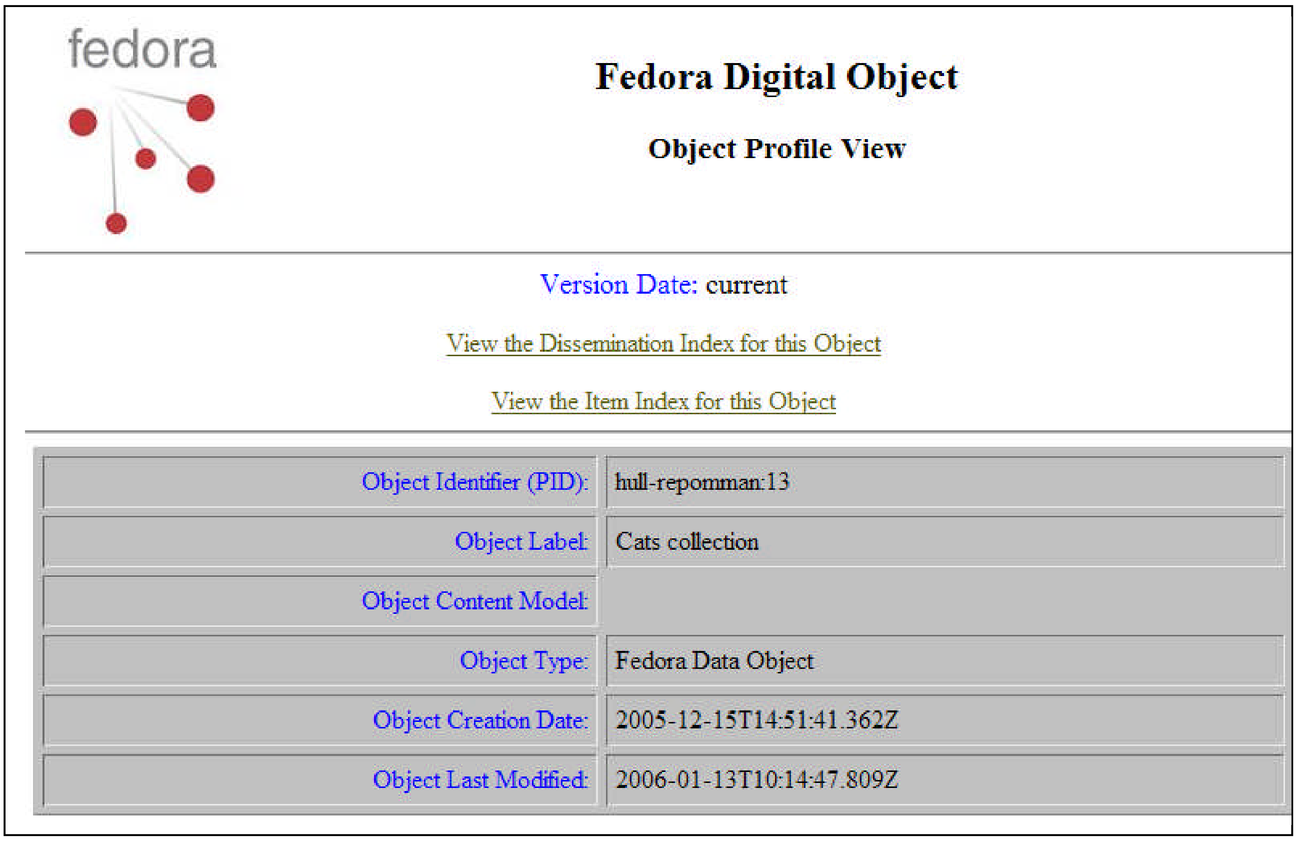
We need now to set up the disseminators - but before we try that, close your collection object and re-open it. Remember that the Fedora admin client requires you to do this before you can see the new datastreams for binding.

- create a new disseminator with an id of DISS1
- whose behaviour is defined by 'demo:Collection (Collection of Objects)'
- using the mechanism 'demo:DualResImageCollection'
- bind 'QUERY' to your 'MemberQuery.txt'
- bind 'LIST' to your 'MemberList.xml', and
- bind 'XSLT' to your 'ViewStylesheet.xsl'

Hopefully, all is now done. Our collection has a dissemination method associated with it and the datastreams are there for the dissemination to use. To see the effect, you need to search for your collection object using a browser

http://*yourBaseURL:port*/fedora/search

and have a look at its dissemination methods. My cats collection object comes up as:



Fedora Digital Object

Object Profile View

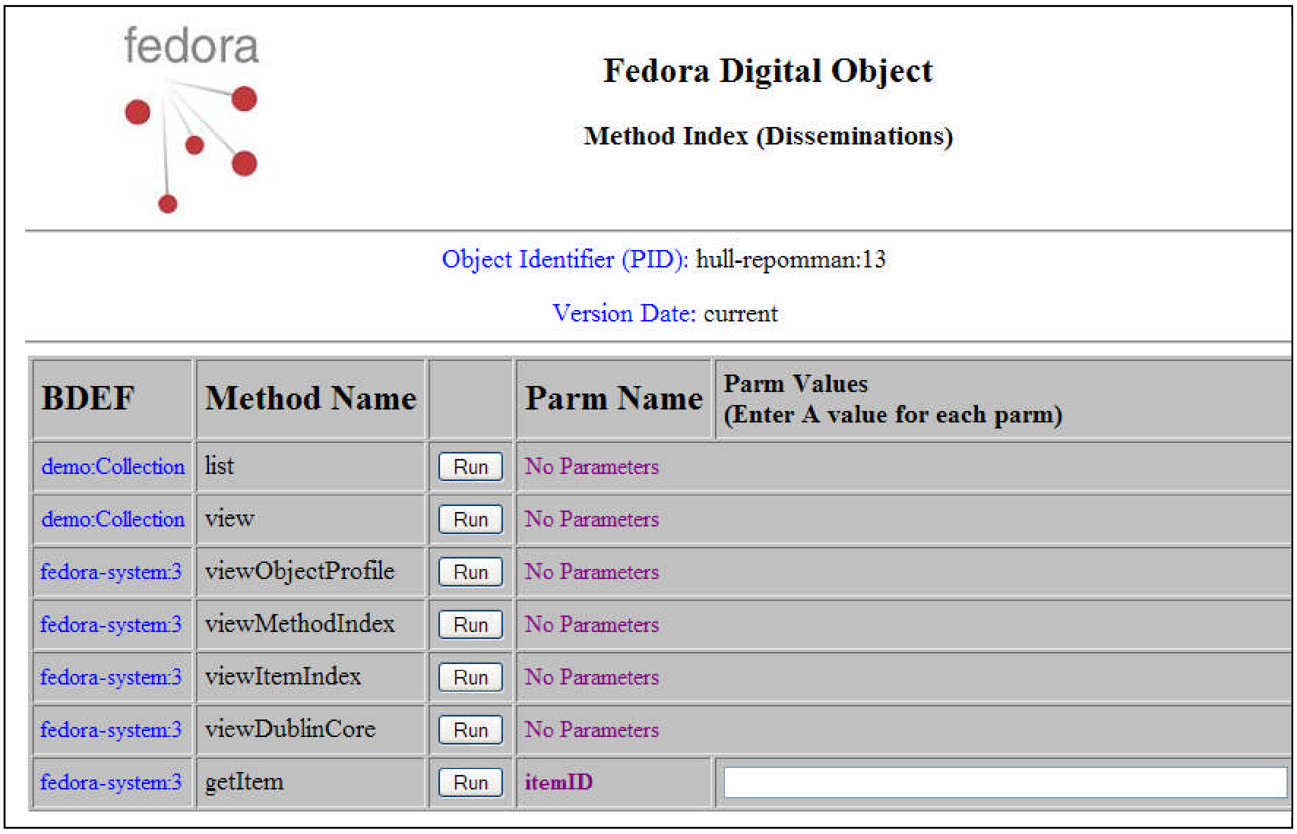
Version Date: current

View the Dissemination Index for this Object

View the Item Index for this Object

Object Identifier (PID):	hull-repomman:13
Object Label:	Cats collection
Object Content Model:	
Object Type:	Fedora Data Object
Object Creation Date:	2005-12-15T14:51:41.362Z
Object Last Modified:	2006-01-13T10:14:47.809Z

We're interested in the dissemination index:



Fedora Digital Object

Method Index (Disseminations)

Object Identifier (PID): hull-repomman:13

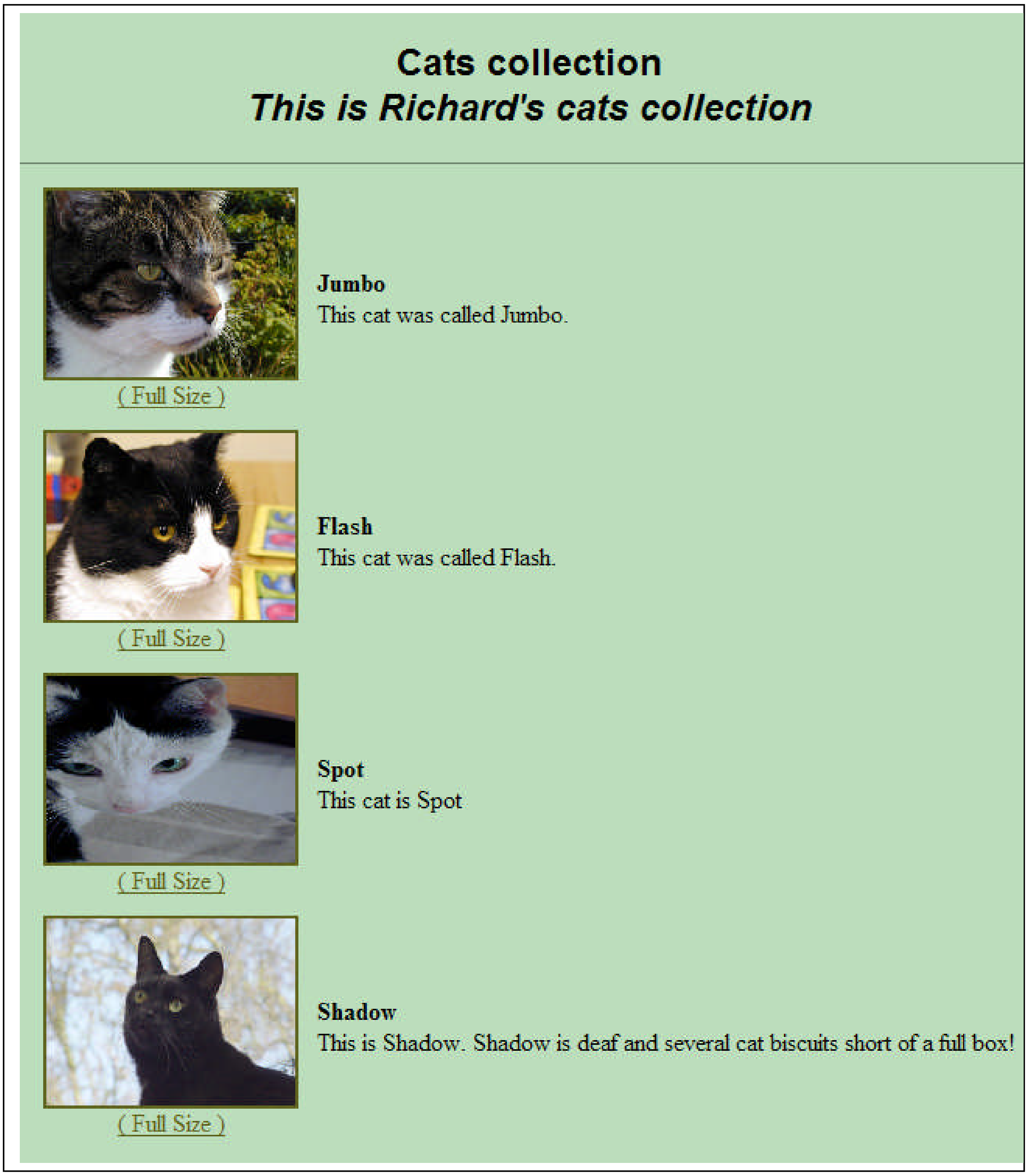
Version Date: current

BDEF	Method Name		Parm Name	Parm Values (Enter A value for each parm)
demo:Collection	list	Run	No Parameters	
demo:Collection	view	Run	No Parameters	
fedora-system:3	viewObjectProfile	Run	No Parameters	
fedora-system:3	viewMethodIndex	Run	No Parameters	
fedora-system:3	viewItemIndex	Run	No Parameters	
fedora-system:3	viewDublinCore	Run	No Parameters	
fedora-system:3	getItem	Run	itemID	

If you first of all run the 'list' method then you should get an XML file that is the result of running your RDF query. Mine looked like this:

```
<sparql xmlns="http://www.w3.org/2001/sw/DataAccess/rf1/result">
  <head>
    <variable name="collTitle"/>
    <variable name="collDesc"/>
    <variable name="member"/>
    <variable name="memberTitle"/>
    <variable name="memberDesc"/>
  </head>
  <results>
    <result>
      <collTitle>Cats collection</collTitle>
      <collDesc>This is Richard's cats collection</collDesc>
      <member uri="info:fedora/hull-repomman:9"/>
      <memberTitle>Jumbo</memberTitle>
      <memberDesc>This cat was called Jumbo.</memberDesc>
    </result>
    <result>
      <collTitle>Cats collection</collTitle>
      <collDesc>This is Richard's cats collection</collDesc>
      <member uri="info:fedora/hull-repomman:10"/>
      <memberTitle>Flash</memberTitle>
      <memberDesc>This cat was called Flash.</memberDesc>
    </result>
    <result>
      <collTitle>Cats collection</collTitle>
      <collDesc>This is Richard's cats collection</collDesc>
      <member uri="info:fedora/hull-repomman:11"/>
      <memberTitle>Spot</memberTitle>
      <memberDesc>This cat is Spot</memberDesc>
    </result>
    <result>
      <collTitle>Cats collection</collTitle>
      <collDesc>This is Richard's cats collection</collDesc>
      <member uri="info:fedora/hull-repomman:12"/>
      <memberTitle>Shadow</memberTitle>
      <memberDesc>This is Shadow.  Shadow is deaf and several cat biscuits short of a full
                        box!</memberDesc>
    </result>
  </results>
</sparql>
```

Running the 'view' method should give you an HTML page the likes of:



Cats collection

This is Richard's cats collection

(Full Size)

Jumbo

This cat was called Jumbo.

(Full Size)

Flash

This cat was called Flash.

(Full Size)

Spot

This cat is Spot

(Full Size)

Shadow

This is Shadow. Shadow is deaf and several cat biscuits short of a full box!

If you have looked closely, the thumbnail image of 'Spot' (who is, at the moment, snoring gently not very far from my desk) is distorted. The ViewStylesheet.xsl file uses image width and height attributes in the HTML to impose a size of 160x120 pixels. This doesn't work too well if you have a portrait-orientated image.

Collecting collections

The work that we are doing with the RepoMMan project will, in time, mean that we need to collect collections. Within Fedora this should be straightforward and it was what I tried next. First of all, I decided that I needed to collect two collections together of the same type. This meant that I had to create another small collection for which I chose an implicit collection of holiday photos taken in and around Barcelona in Spain.

Taking my two implicit collection objects in turn, I made them each a member of a new collection object 'hull-repomman:19' by adding to them a RELS-EXT datastream asserting their membership. The datastream for hull-repomman:18 reads:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rel="info:fedora/fedora-system:def/relations-external#">
  <rdf:Description rdf:about="info:fedora/hull-repomman:18">
    <rel:isMemberOf rdf:resource="info:fedora/hull-repomman:19"/>
  </rdf:Description>
</rdf:RDF>
```

The new collection object 'hull-repomman:19' was given all the appropriate datastreams and dissemination information to deal with dual resolution images.

My RDF query read:

```
select $collTitle $collDesc $subcollection $member $memberTitle $memberDesc
from   <#ri>
where
       <info:fedora/hull-repomman:19> <dc:title>                    $collTitle
and    <info:fedora/hull-repomman:19> <dc:description>              $collDesc
and    $subcollection                  <fedora-rels-ext:isMemberOf>
                                                   <info:fedora/hull-repomman:19>
and    $member                         <fedora-rels-ext:isMemberOf> $subcollection
and    $member                         <dc:title>                    $memberTitle
and    $member                         <dc:description>              $memberDesc
```

This worked fine, listing the contents of both implicit collections. Slightly surprised at my own success, I went for the top prize - listing all the collections, implicit and explicit. I simply modified the RDF query:

```
select $collTitle $collDesc $subcollection $member $memberTitle $memberDesc
from   <#ri>
where
(
(        <info:fedora/hull-repomman:19> <dc:title>                        $collTitle
and      <info:fedora/hull-repomman:19> <dc:description>                  $collDesc
and      $subcollection                 <fedora-rels-ext:isMemberOf>
                                                        <info:fedora/hull-
repomman:19>
and      $member                        <fedora-rels-ext:isMemberOf> $subcollection
and      $member                        <dc:title>                        $memberTitle
and      $member                        <dc:description>                  $memberDesc)
or
(        <info:fedora/hull-repomman:19> <dc:title>                        $collTitle
and      <info:fedora/hull-repomman:19> <dc:description>                  $collDesc
and      $subcollection                 <fedora-rels-ext:isMemberOf>
                                                        <info:fedora/hull-repomman:19>
and      $subcollection                 <fedora-rels-ext:hasMember>  $member
and      $member                        <dc:title>                        $memberTitle
and      $member                        <dc:description>                  $memberDesc)
)
```

... and was pleased to see that it worked. So Fedora's RDF queries seem to support brackets and logical ORs.

Dublin Core metadata

Way back we said that we would return to the Dublin Core metadata that could legitimately be put into the DC datastream.

The list supported internally by Fedora is:

- title
- creator
- subject
- description
- publisher
- contributor
- date
- type
- format
- identifier
- source
- language
- relation
- coverage
- right

Up to version 2.0 of Fedora, qualified Dublin Core is not supported here. In fact this datastream was intended for administrative use rather than for OAI harvesting.

External relationships

Fedora's external relationship definitions cover the following, some of which we have already used. All but the last are mirrored pairings so that either explicit or implicit relationships can be declared.

- is part of / has part
- is constituent of / has constituent
    - this is a refinement of the 'part' relationship which implies referential integrity. The understanding is that the part cannot stand alone in any meaningful way
- is member of / has member
    - this is effectively a set relationship; a member of a set can reasonably be used on its own
- is subset of / has subset
- is member of collection / has collection member
    - this level of relationship was intended to represent the notion of a high level digital collection
- is derivation of / has derivation
- is dependent of / has dependent
- is description of / has description
- is metadata for / has metadata
- is annotation of / has annotation
- has equivalent

For the sake of legibility I have inserted a lot of spaces in the list. Don't forget that in a RDF-EXT datastream, use of these relationships would take the form:

```
<rel:isMemberOf rdf:resource="info:fedora/hull-repomman:13"/>
```

The relationship term does not have spaces and conventionally has upper case letters to denote the beginnings of words.

Two more types of object (an external and a redirect object)

Rather a lot of pages ago we set about creating a first object which was placed in the Fedora control group 'managed content', since then we have created many more. But what about the other two control groups available for an object that is more than just metadata? Let's look at them now.

Externally referenced content

Externally referenced content is content that is not held directly by Fedora but which resides somewhere else and can be referenced through a URL. Metadata about it is held by the local Fedora installation and thus it can appear when the repository is searched. If the user causes the object to be retrieved it is done in such a way that they will be unaware that it has come from a source remote to the repository.

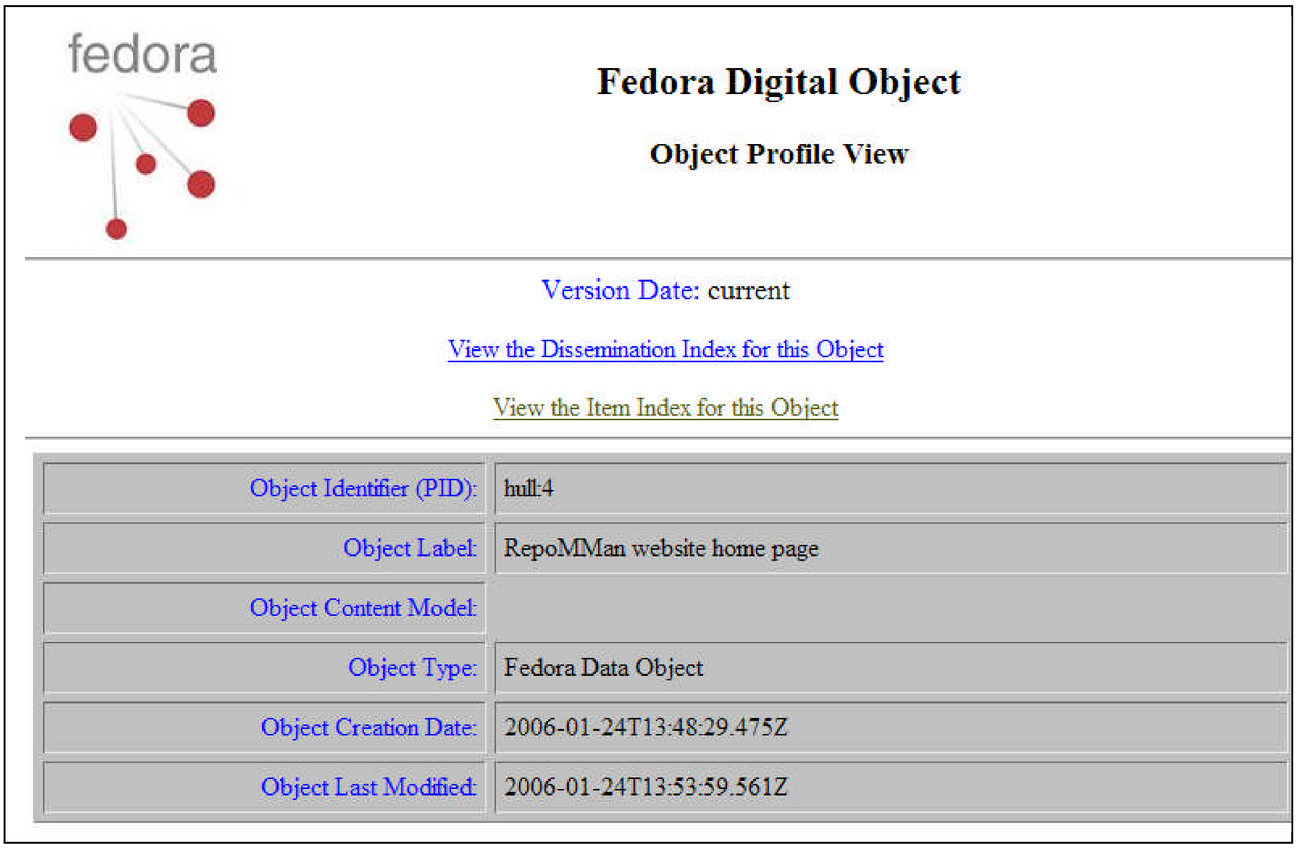
As an example of how to do this, I chose to use the RepoMMan Project's home page as externally referenced content. Why this was not actually the best of ides will emerge later.

The basic object was created in the usual way and a label assigned to it. On the 'datastreams' record I set up a 'new' datastream with an ID of 'HTMLLink' and set the control group to 'externally referenced content'. As this was a link to an HTML page, the MIME type was set to 'text/html' and the location was duly filled in:

http://www.hull.ac.uk/esig/repomman/index.html

And that was essentially that. In fact I went on to edit a couple of items into the 'DC' datastream (description and rights) but that was not a necessary part of the process.

Using the Fedora web search tool, I looked for my object:



fedora

Fedora Digital Object

Object Profile View

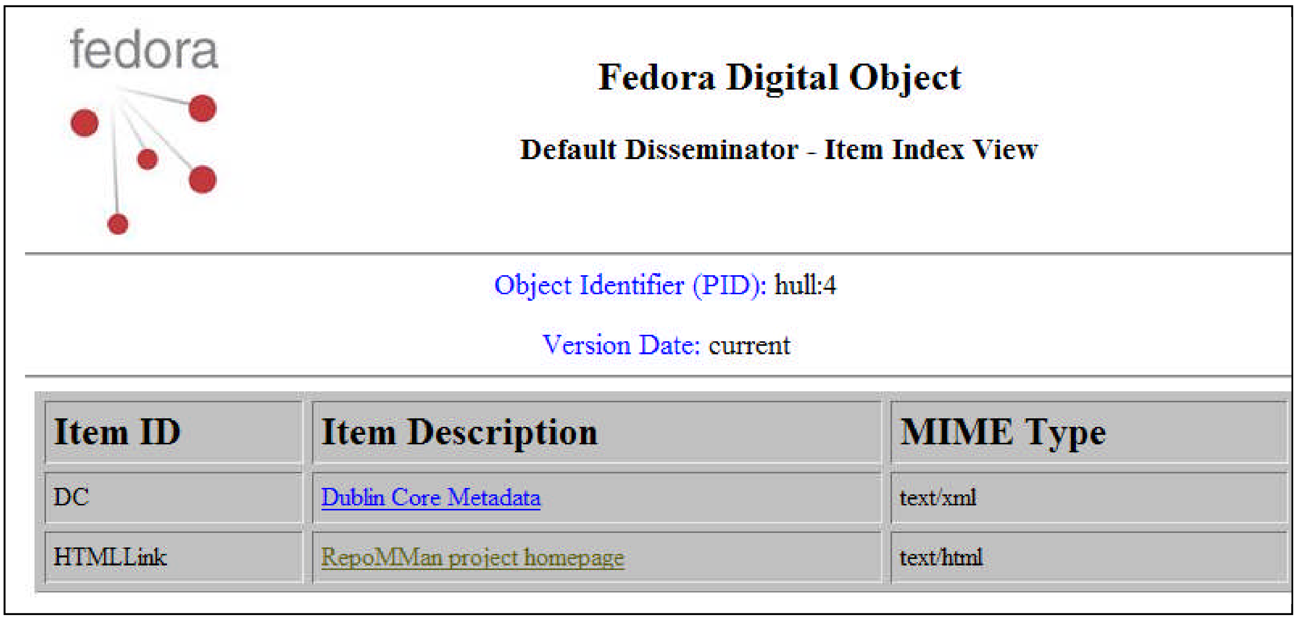
Version Date: current

View the Dissemination Index for this Object

View the Item Index for this Object

Object Identifier (PID):	hull:4
Object Label:	RepoMMan website home page
Object Content Model:	
Object Type:	Fedora Data Object
Object Creation Date:	2006-01-24T13:48:29.475Z
Object Last Modified:	2006-01-24T13:53:59.561Z

The dissemination index covers the standard Fedora behaviours and no more because we have not added any disseminators of our own. The item index lists the datastreams:



fedora

Fedora Digital Object

Default Disseminator - Item Index View

Object Identifier (PID): hull:4

Version Date: current

Item ID	Item Description	MIME Type
DC	Dublin Core Metadata	text/xml
HTMLLink	RepoMMan project homepage	text/html

Clicking the link for the 'RepoMMan project homepage' brings up that very page in my browser. But it is important that we realise how it got there. When I looked at the URL used I found:

http://150.237.xxx.xxx:8080/fedora/get/hull:4/HTMLLink

Not surprising, you might think. We said that the page would be retrieved so that the user couldn't tell that it came from outside the repository - and so it is. You may recall my saying that this web page was perhaps not the best one to choose as an example of an external object - and this is why. RepoMMan's website uses relative references within links. If I try and navigate to the index of the documents section, the link says go to 'documents/index.html' meaning, of course:

http:www.hull.ac.uk/esig/repomman/documents/index.html

What happens instead here is that the link directs to

http://150.237.xxx.xxx:8080/fedora/get/hull:4/documents/index.html

resulting in a messy screen of error messages. Lesson? Don't use 'externally referenced content' for a website that uses relative links. It would have been much better to use the 'redirect' control group. For all that, we have demonstrated the principal.

Redirected content

The 'redirect' control group is for content stored external to the Fedora repository but of such a form that Fedora cannot manage it internally. We have just seen one example, a website using relative links, another might be streamed media. If the object is called, Fedora simply redirects the user to the original - but, of course, Fedora can still hold metadata about it. It just so happens that I have access to a Real Media server at the University of Hull that was set up for a foreign language learning project some years ago. I decided to create an object for a piece of streamed video. (If you are going to follow this one through with me you will need to ensure that Real Player is installed on your computer.)

Again, I set up a basic object with a useful label. I then added a datastream called 'RMVideo' with a 'redirect' content model. I puzzled for a while over the MIME type because Fedora 'out of the box' does not provide anything for multimedia. In the end I settled for 'text/plain' and that worked for the purposes of a quick test. The URL is:

http://150.237.4.70:8080/ramgen/paris/intros/paris990126.rm

(Sorry about that!). Once the object is saved, you should be able to find it via the search tool and see that the item index contains a link for the 'RMVideo' datastream. Clicking it should launch your local Real Player and deliver a short introduction to Paris - it's a chance to brush up on your French! If you have no French, well the pictures are pretty.

Behaviour Definitions (BDefs) and Behaviour Mechanisms (BMechs)

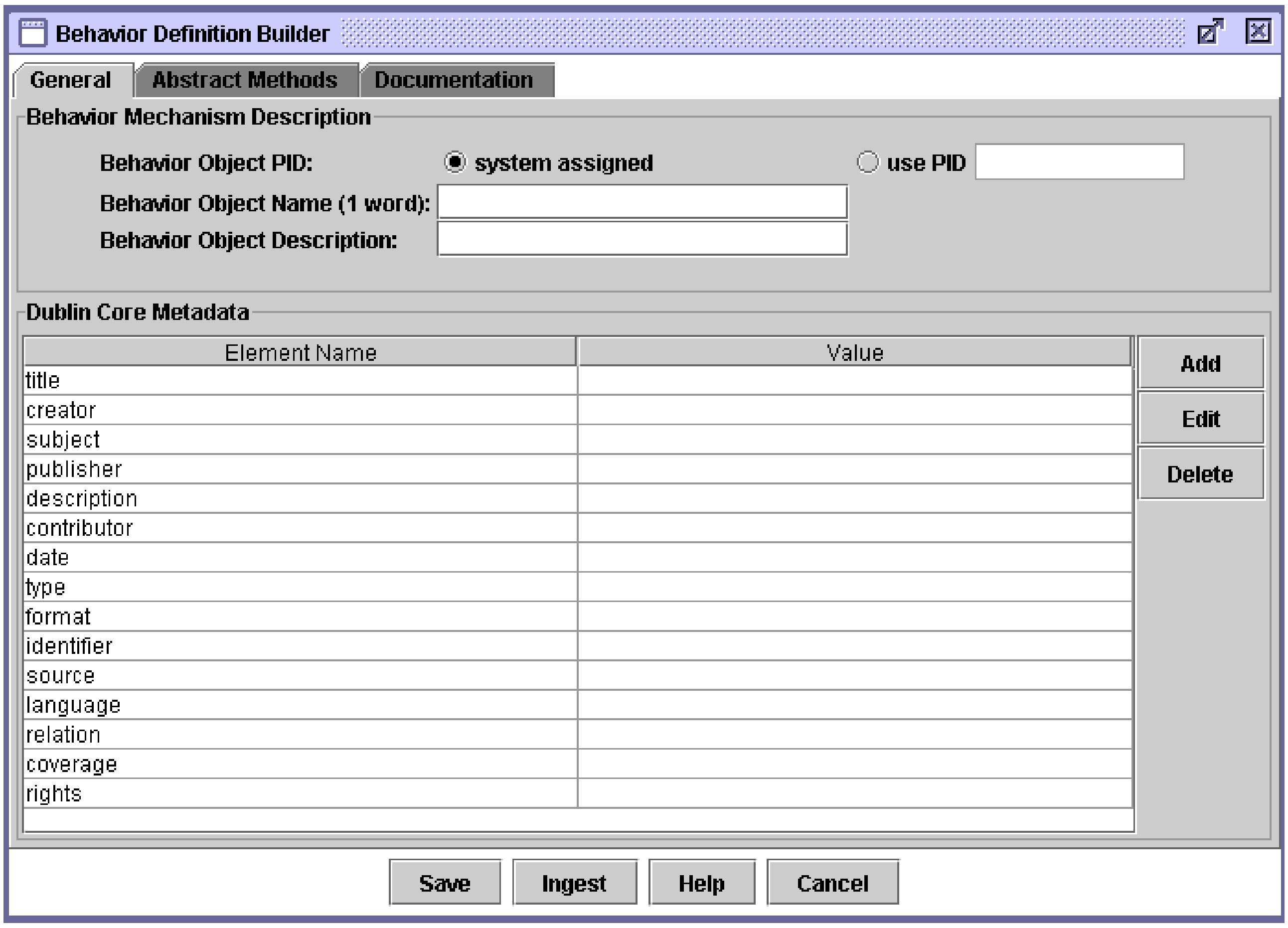
We've had a good look around straightforward Fedora objects, now is the time to have a look in more detail at Behaviour Definitions (BDefs) and Behaviour Mechanisms (BMechs). Thus far, we have used only the demonstration BDefs and BMechs provided in the Fedora download. Let's have a look at building a pair of our own. The Fedora admin client provides tools for doing this.

For the sake of an example we are going to create a disseminator which deals with a multi-datastream image object. In my case the object contained four image datastreams, one a full-size jpg just as it came off the digital camera, one a corrected, full-size image (perhaps it had been sharpened or had its colour balance corrected), one a "screen-size" object derived from the full-size (this, of course, depends what you think screen-size should be. I went for 1024 x 768), and one a thumbnail-size image (say 120x90). If you think that this is an odd collection of datastreams, it is based on our first attempt at defining a standard image content model for our repository. There is an archive datastream (never to be used by the public) and a choice of three derived images which can be used as we feel appropriate. In addition to the image datastreams we will add a 'UoHMetadata' datastream with the metadata that is intended for public exposure. The disseminator will allow delivery of any of the three public image datastreams.

You may think that this is an odd use for a disseminator! Serving up three datastreams that could be accessed directly... Yes and no. We intend implementing a policy that allows public (anonymous) users to access objects only via disseminators and to have no direct access to any underlying datastreams - thus we need a disseminator to cope with the three main image datastreams. The archive copy will be forever hidden from them.

The first job, then, if you are going to follow this through with me is to create yourself a new object. It needs you to add, as a minimum, three datastreams called 'fullsize', 'screensize' and 'thumbnail' or similar. When you've done that, set it to one side until we have the disseminator set up.

Now we need to define the behaviours that we want. In the admin client, select 'File | New | Behaviour definition'. You should get a screen like this:



Behavior Mechanism Description

Behavior Object PID: [✓ system assigned] use PID

Behavior Object Name (1 word):

Behavior Object Description:

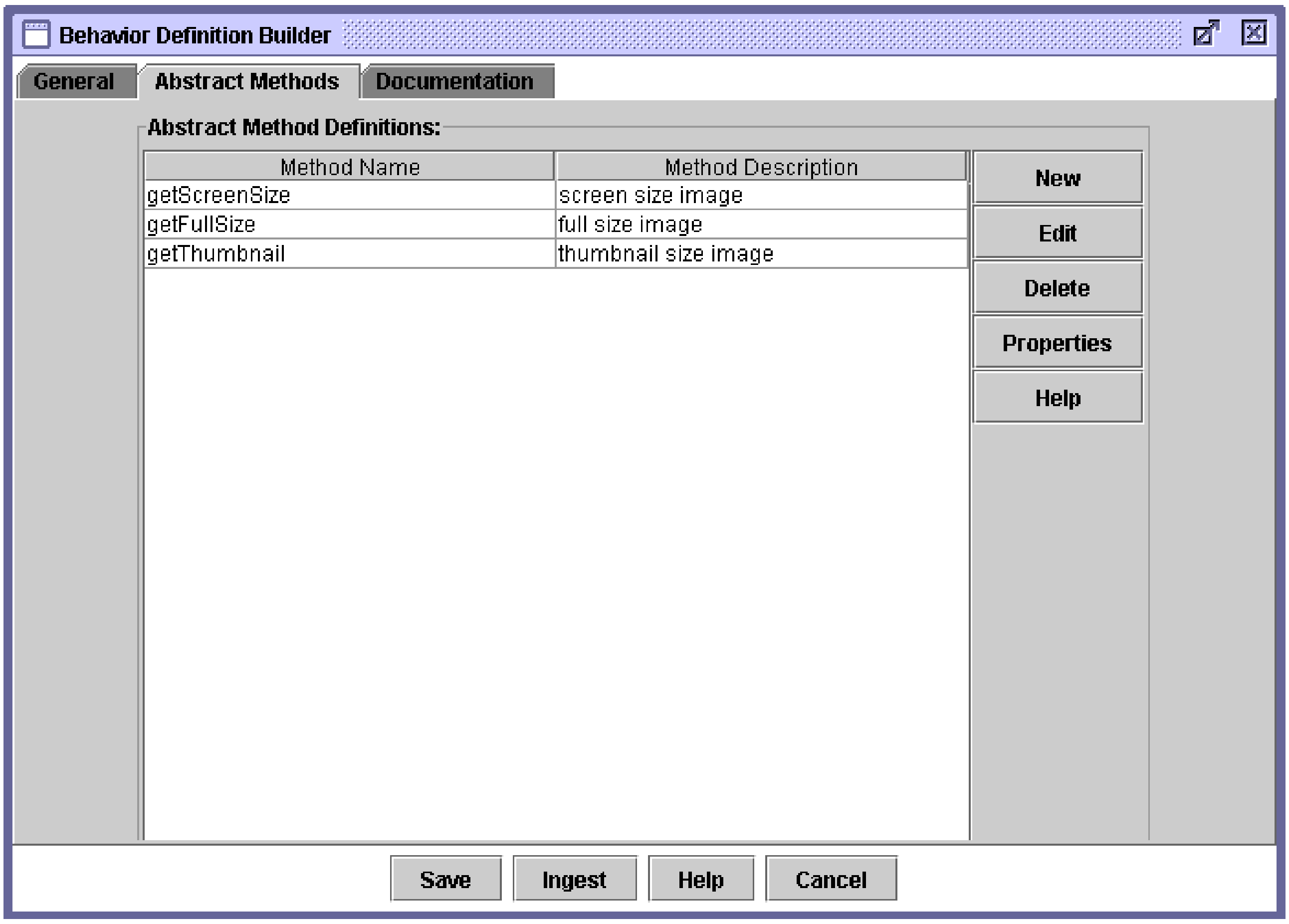
Dublin Core Metadata

Element Name	Value
title	
creator	
subject	
publisher	
description	
contributor	
date	
type	
format	
identifier	
source	
language	
relation	
coverage	
rights	

Fill in a name for it (maybe threeRes? This would be used as a WSDL service name, for instance) and give it a description (mine was 'UoH Standard Image Model behaviour definition'). Fill in as much of the internal metadata as you think fit (a title at the very least).

Now move to the 'Abstract Methods' tab an define the three methods you will need using the 'New' button:

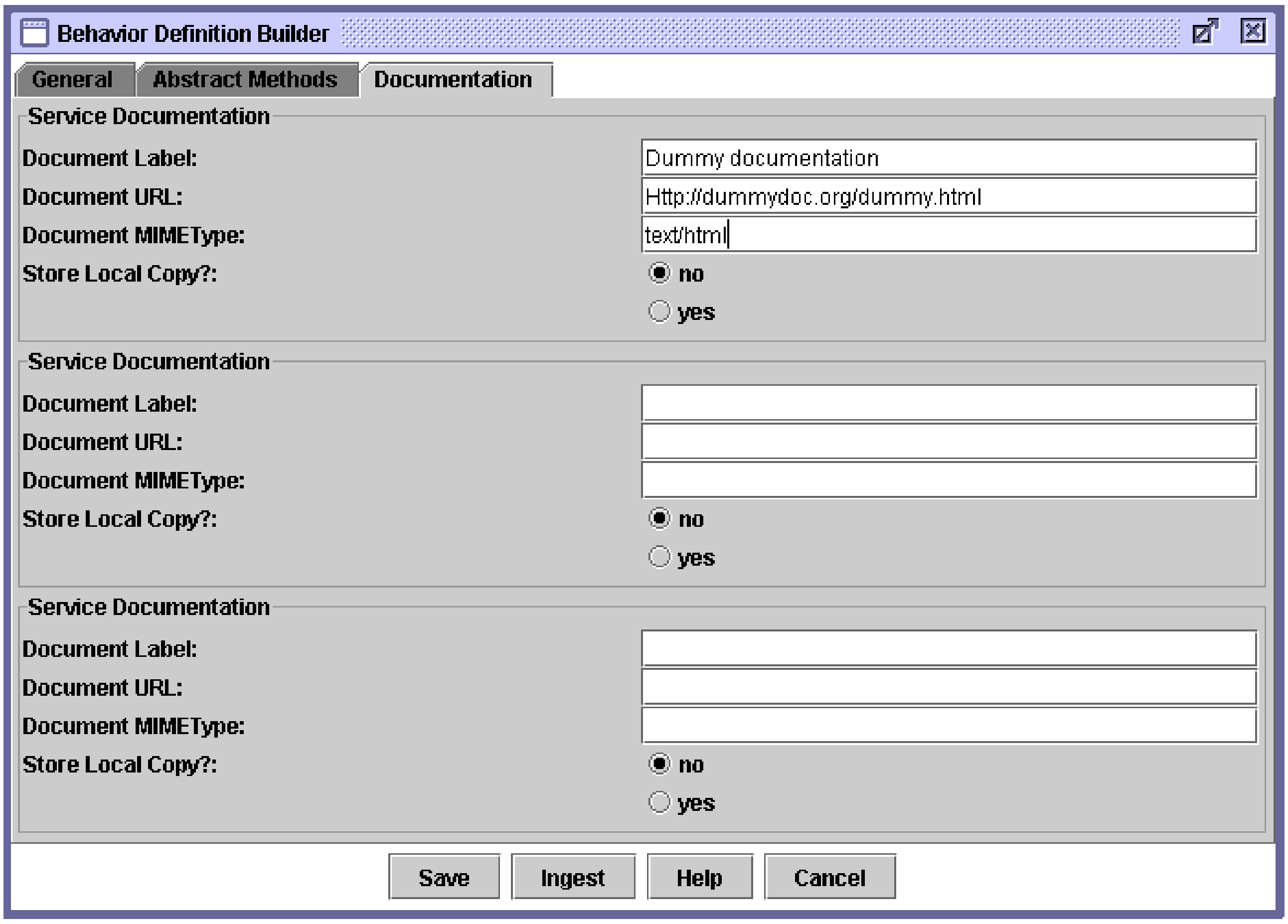
- getScreenSize - screen size image
- getFullSize - original size image
- getThumbnail - thumbnail size image



Method Name	Method Description
getScreenSize	screen size image
getFullSize	full size image
getThumbnail	thumbnail size image

If any of our methods had required parameters we could now select it and, using the 'properties' button set up such things as defaults. In this case we can skip that stage.

In the real world, we would want to add some documentation for this object - and Fedora makes this compulsory. However, for the moment, we can go to the 'Documentation tab and fill in some dummy information:

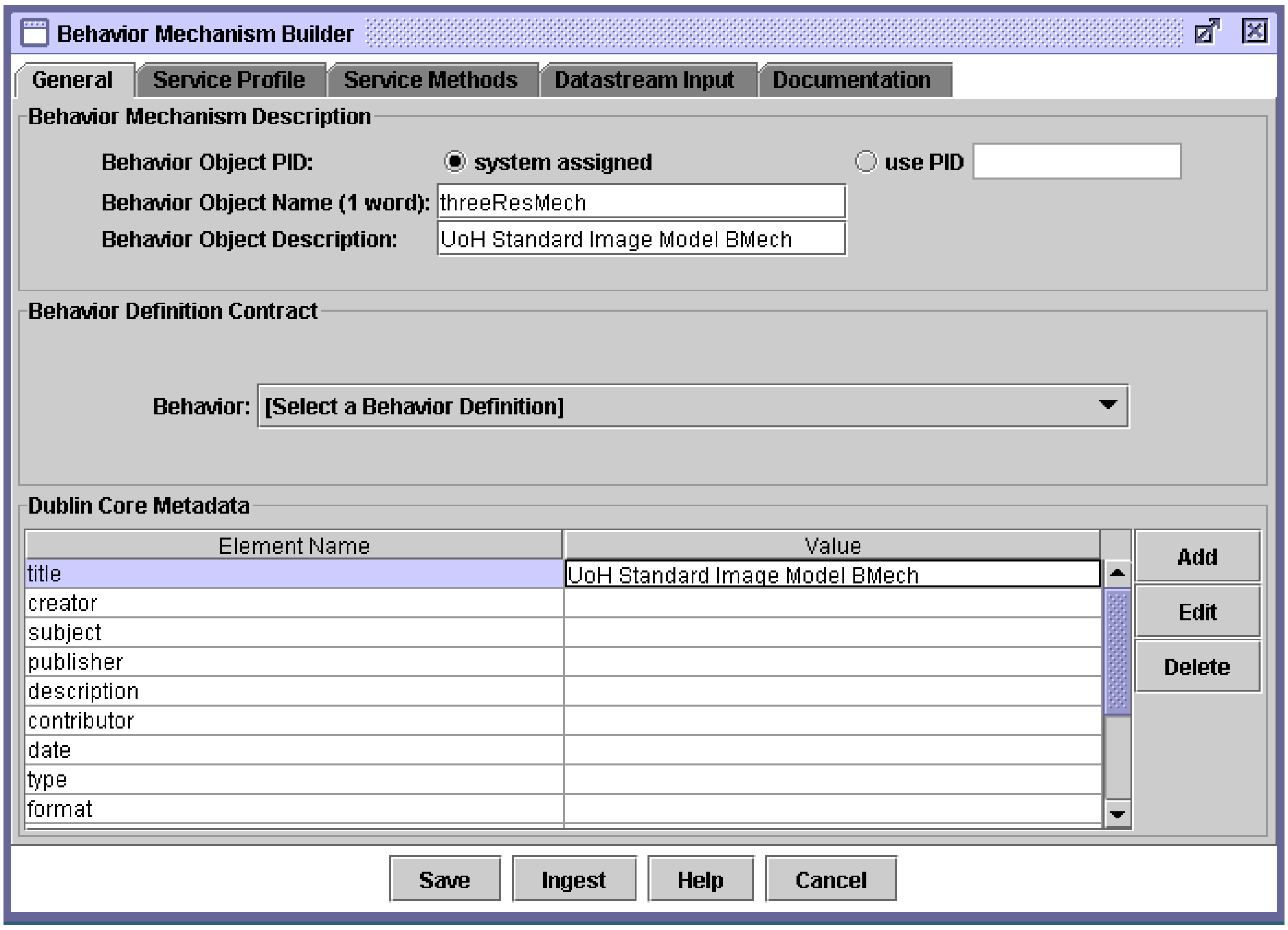


Service Documentation	
Document Label:	Dummy documentation
Document URL:	Http://dummydoc.org/dummy.html
Document MIMEType:	text/html
Store Local Copy?:	[✓ no] yes
Service Documentation	
Document Label:	
Document URL:	
Document MIMEType:	
Store Local Copy?:	[✓ no] yes
Service Documentation	
Document Label:	
Document URL:	
Document MIMEType:	
Store Local Copy?:	[✓ no] yes

Now we can go back to the 'General' tab and ingest the object - well, almost. When I did this (with Fedora 2.0) and clicked 'Ingest' my admin client generated an error. I first had to click 'use PID' and enter a PID name (BDef:1). The BDef then saved but was given a system-assigned PID (hull:10) anyway! (Fedora 2.1 seems to have fixed this problem.)

Right: on to the BMech.

In the admin client select 'File | New | Behaviour mechanism' and fill in the basics:



Behavior Mechanism Description

Behavior Object PID: (● system assigned) (○ use PID)

Behavior Object Name (1 word): threeResMech

Behavior Object Description: UoH Standard Image Model BMech

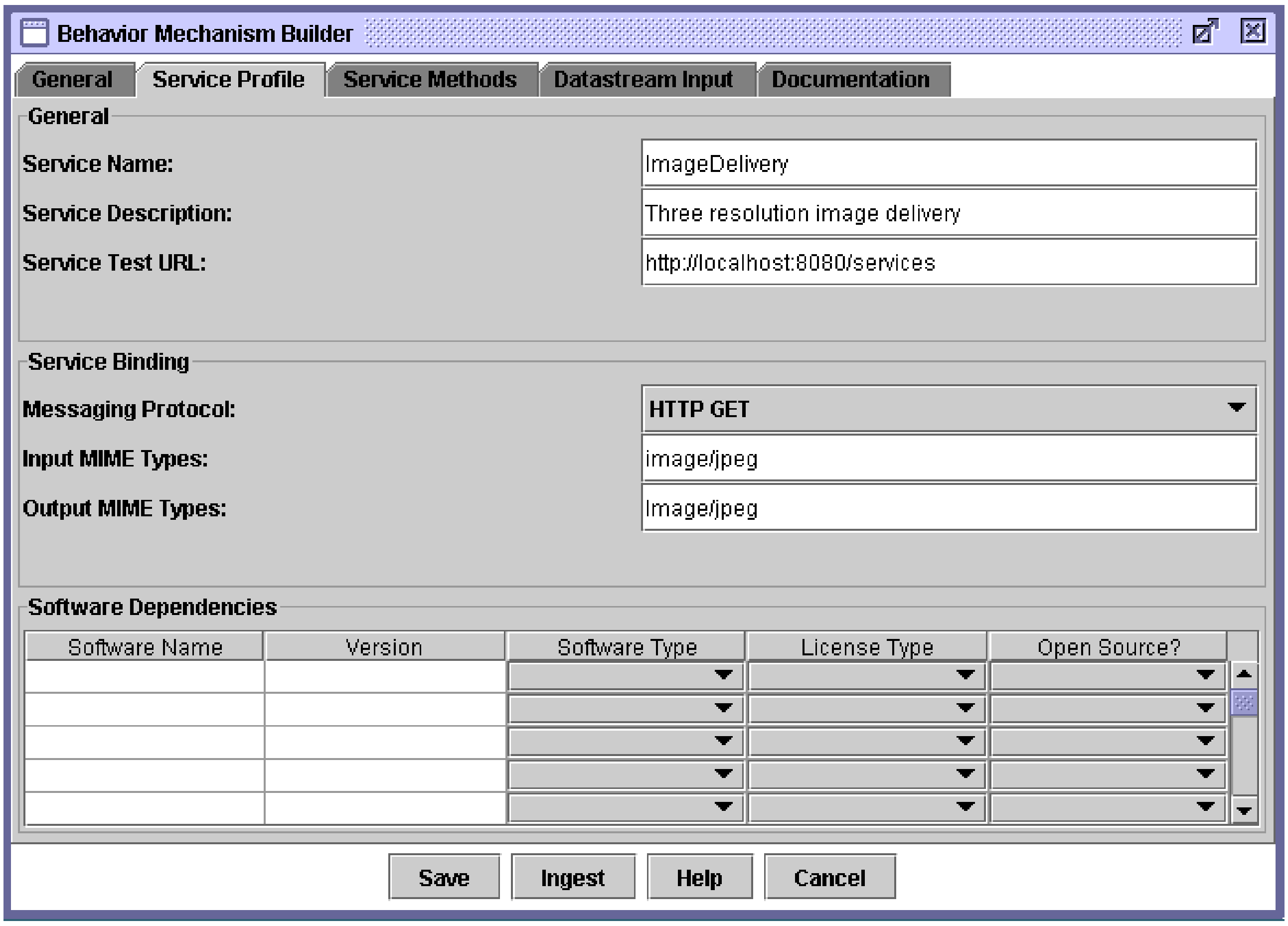
Behavior Definition Contract

Behavior: [Select a Behavior Definition]

Dublin Core Metadata

Element Name	Value
title	UoH Standard Image Model BMech
creator	
subject	
publisher	
description	
contributor	
date	
type	
format	

Pull down the 'Behaviour' List and select the behaviour definition that you have just created. Now switch to the 'Service Profile' tab. This is for optional service metadata; on this occasion we can just fill it with dummy values:



General

Service Name:	ImageDelivery
Service Description:	Three resolution image delivery
Service Test URL:	http://localhost:8080/services

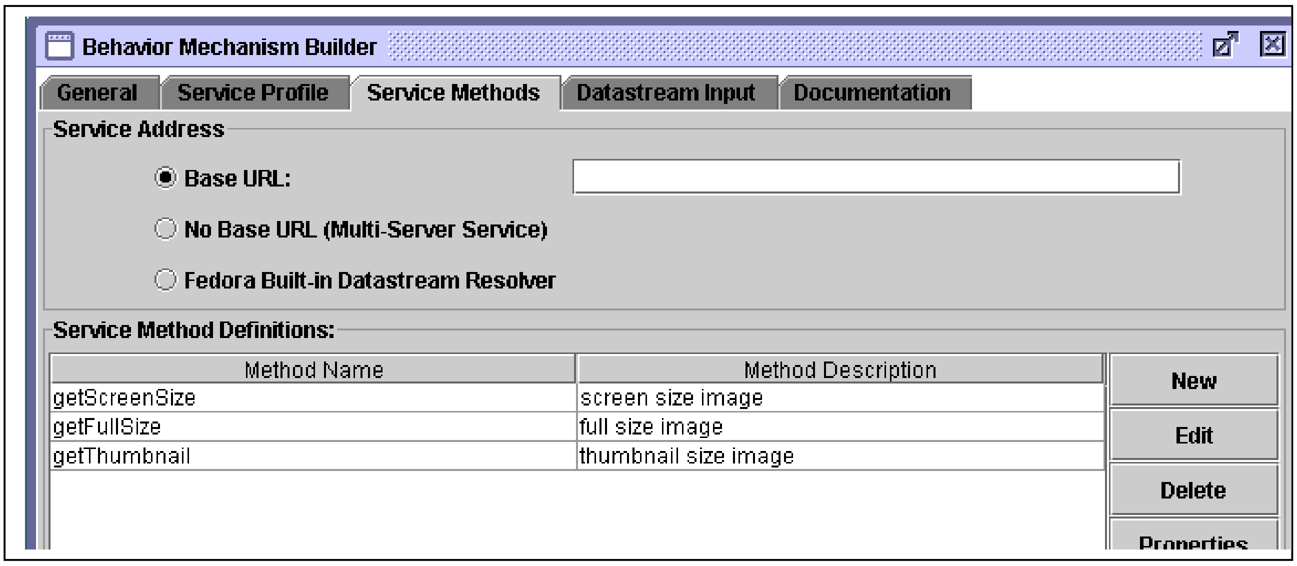
Service Binding

Messaging Protocol:	HTTP GET
Input MIME Types:	image/jpeg
Output MIME Types:	Image/jpeg

Software Dependencies

Software Name	Version	Software Type	License Type	Open Source?

Now switch to the 'Service Methods' tab where you should see listed the three methods that you defined in the BDef:



Service Address

● Base URL:

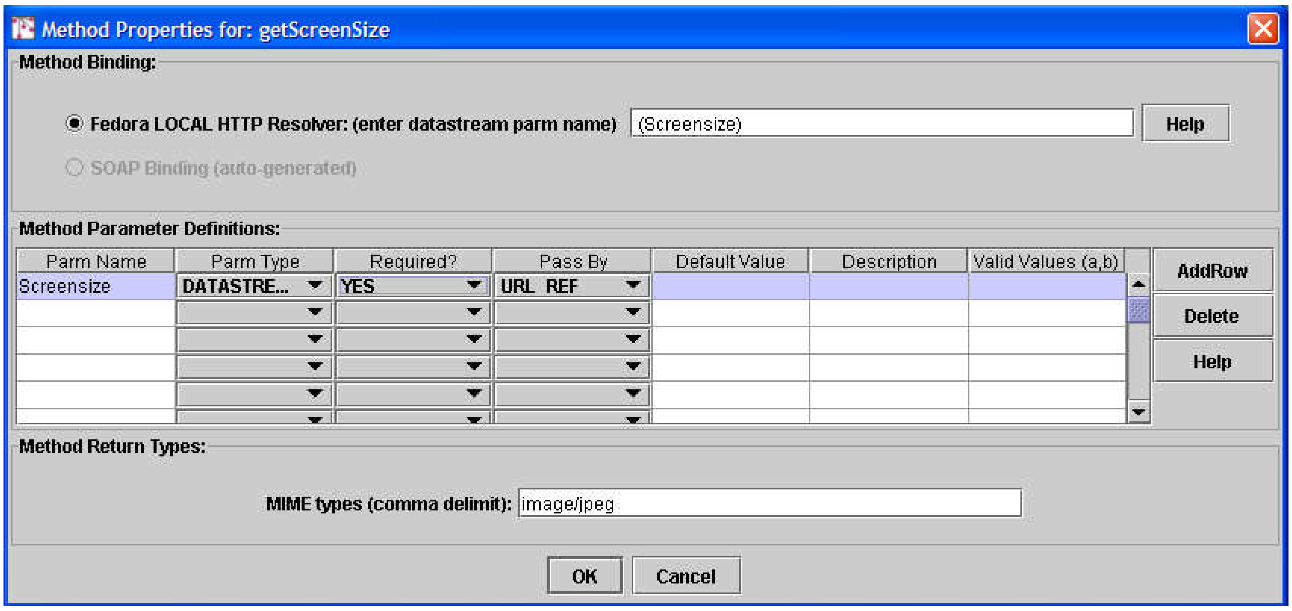
○ No Base URL (Multi-Server Service)

○ Fedora Built-in Datastream Resolver

Service Method Definitions:

Method Name	Method Description
getScreenSize	screen size image
getFullSize	full size image
getThumbnail	thumbnail size image

At the top of the page, click to use the 'Fedora Built-in Datastream Resolver. Click on the first method and choose to set its properties:


Method Properties for: getScreenSize
Method Binding:
Fedora LOCAL HTTP Resolver: (enter datastream parm name)
SOAP Binding (auto-generated)
Help
Method Parameter Definitions:
AddRow
Delete
Help
Method Return Types:
OK
Cancel

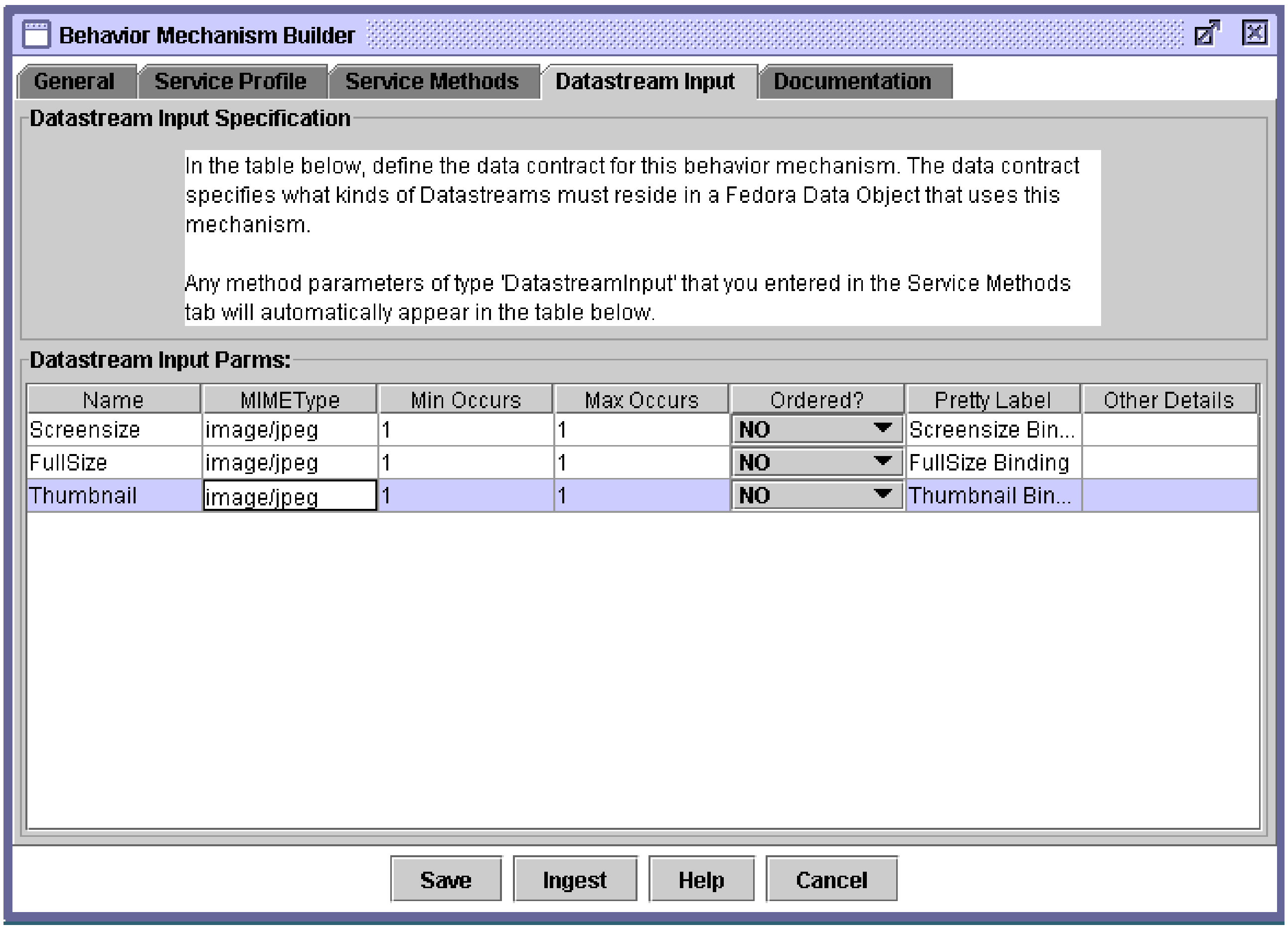
● Fedora LOCAL HTTP Resolver: (enter datastream parm name) (Screensize)

Parm Name	Parm Type	Required?	Pass By	Default Value	Description	Valid Values (a,b)
Screensize	DATASTRE...	YES	URL REF			

MIME types (comma delimit): image/jpeg

Enter a datastream parameter name at the top in (round brackets). The same name in the list below, define the type as a 'Datastream', say that it is required, and set the 'Pass By' as 'URL REF'. Set the MIME type at the bottom of the page.

Now move to the 'Datastream Input' tab where everything necessary has already been filled in for you except the MIME types ('image/jpeg'). You need to do that.



In the table below, define the data contract for this behavior mechanism. The data contract specifies what kinds of Datastreams must reside in a Fedora Data Object that uses this mechanism.

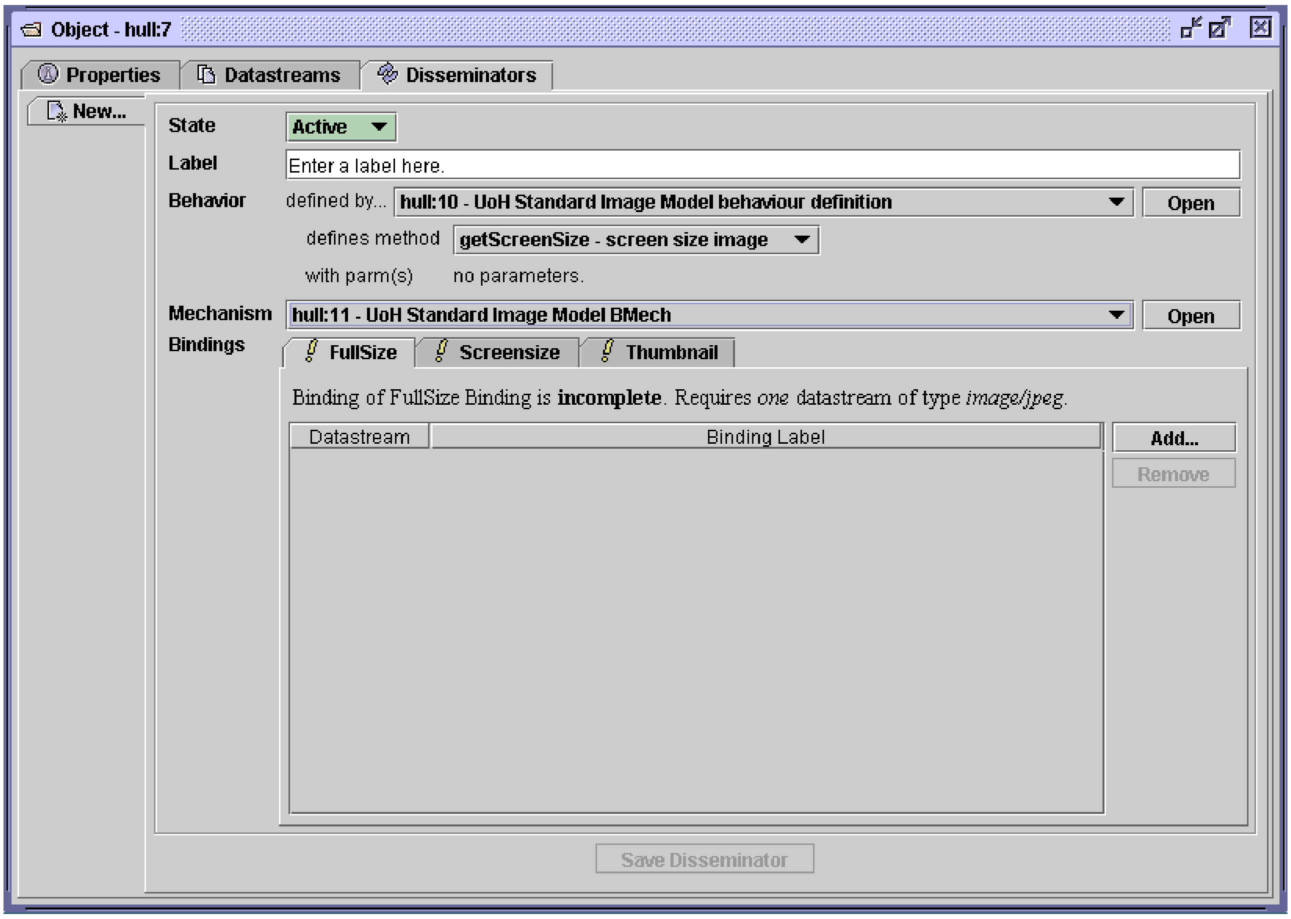
Any method parameters of type 'DatastreamInput' that you entered in the Service Methods tab will automatically appear in the table below.

Name	MIMEType	Min Occurs	Max Occurs	Ordered?	Pretty Label	Other Details
Screensize	image/jpeg	1	1	NO	Screensize Bin...	
FullSize	image/jpeg	1	1	NO	FullSize Binding	
Thumbnail	image/jpeg	1	1	NO	Thumbnail Bin...	

And finally, the 'Documentation' tab. Fedora requires this, but for now fill it with dummy data as you did for the BDef.

Now go back to the 'General' tab and try to ingest the BMech. You will probably have the same problem with the PID again. I had to supply a PID of BMech:1 and then got a new PID of Hull:11. (But as I said, this seems to be fixed in Fedora 2.1)

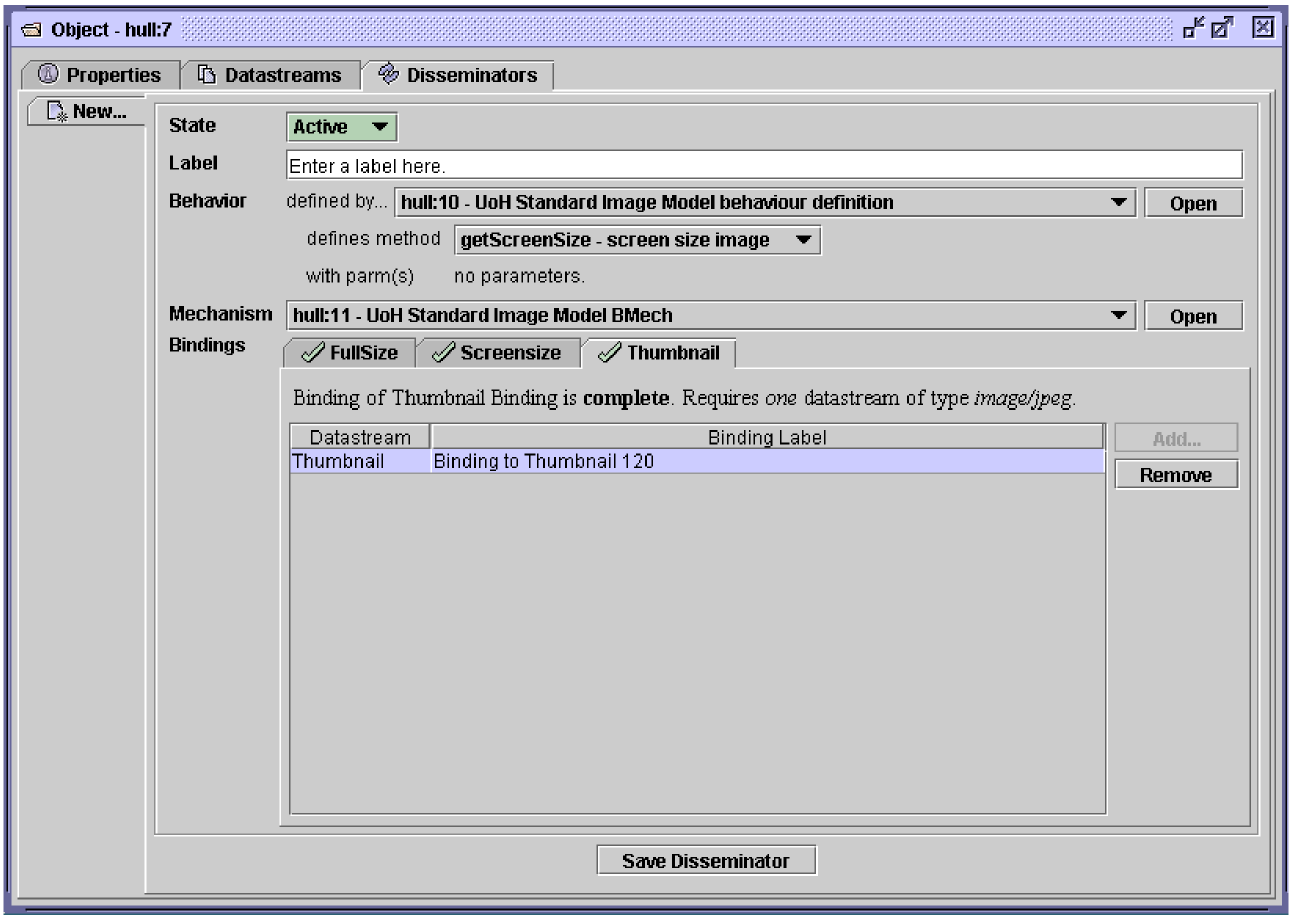
OK. Now go back to your image object and go to the 'Disseminators' tab. Select the behaviour definition object that you want and then the corresponding mechanism:



State: Active
Label: Enter a label here.
Behavior: defined by... hull:10 - UoH Standard Image Model behaviour definition
defines method getScreenSize - screen size image
with parm(s) no parameters.
Mechanism: hull:11 - UoH Standard Image Model BMech
Bindings

Binding of FullSize Binding is **incomplete**. Requires *one* datastream of type *image/jpeg*.

Add datastreams to each of the three bindings:



State: Active

Label: Enter a label here.

Behavior defined by... hull:10 - UoH Standard Image Model behaviour definition

defines method getScreenSize - screen size image

with parm(s) no parameters.

Mechanism: hull:11 - UoH Standard Image Model BMech

Bindings: ✓ FullSize | ✓ Screensize | ✓ Thumbnail

Binding of Thumbnail Binding is **complete**. Requires *one* datastream of type *image/jpeg*.

Datastream	Binding Label
Thumbnail	Binding to Thumbnail 120

...and save the disseminator. Now close the object in the editor. But does it work? Let's try it. In my case I need to get object hull:7 using the 'getScreenSize' method defined by object hull:10. In terms of a URL:

```
http://........../fedora/get/hull:7/hull:10/getScreenSize
```

Well mine works! I hope that yours did too.

XACML security

Now that we have a good grasp of Fedora basics, it is perhaps time to look at the new feature introduced in version 2.1: XACML security.

Patches

We were one of the first sites to test the security system in the 2.1 release and we found two problems that you need to know about from the start. Some of the demonstration security scripts are incorrectly structured (which can cause the server to crash at startup), and there is a bug in the Fedora code that can cause the security system to deny access (wrongly) when some scripts are used in combination. These problems have now been identified. If you are working with the Fedora 2.1 download and want to use XACML security you need to get the following new material:

If your Fedora download pre-dates 2006-02-20 you need:

http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-object-policies/demo-11.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-object-policies/demo-26.xml

http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-DC-to-all-users.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-DC-to-tomcat-group-ALT1.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-DC-to-tomcat-group-ALT2.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-MRSID-if-not-tomcat-role.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-TEISOURCE-to-tomcat-user.xml
http://www.fedora.info/download/2.1/userdocs/server/security/xacml-policies/examples/example-repository-policies/apia-tighten-defaults/apia-restrict-datastreams/deny-apia-datastream-all-to-all-users.xml

In all cases of 2,1, you also need the patch identified as Bugzilla #149 on the Fedora website at www.fedora.info.

Tips for getting going

In addition to the patches, there are a few tricks that you need to know about.

(1) It is quite likely that some of the things you will want to do will require you to stop and restart the Fedora server (for instance, changing roles in the Tomcat user file). We had a few problems doing this over a Telnet connection until we realised that the Telnet username and password needed to be the same as the Fedora administrator username and password; if this is not the case the server shutdown stops in some indeterminate state. (User 'root' then needs to do a second shutdown and initiate the restart.) If, despite having the correct username and password, you get a 'permission denied' error somewhere in the restart have a look at the ownership of 'nohup.out'. If the owner is 'root', that is your problem. Delete, or

better - rename, the file and have the Fedora administrator restart the server. Nohup.out will be recreated with the correct ownership, you should check that your Fedora administrator has full rights on the new file.

(2) One thing you should not normally have to do is to restart the server just because you have changed some security policies. There is a

```
fedora-reload-policies
```

command to do this on a running server. On a Windows installation the command is typed just as above. On a non-Windows installation you need to add a protocol argument:

```
fedora-reload-policies http
```

If you do not, you get an error message from the server containing a warning about the necessary syntax. Only the protocol seems to be necessary; the rest of the arguments listed in the message are not.

(3) Finally, a word of warning. Fedora provide a utility

```
validate-policy
```

to check the syntax and structure of an XACML security file before you load it. (The server doesn't respond well to invalid files!) There is a bug in this utility which means that, if you 'validate' a non-existent file, the tool returns 'OK'. This is a real catch because the naming convention suggested by Fedora for policies means that it is *very* easy to mis-type a name. At the time of writing (2006-04-03) this bug has not been fixed. You just have to be *very* careful using the tool.

Backend security considerations

You need to be aware that two issues can arise if Fedora makes calls of its own which invoke the security system. This can arise, for instance, when one disseminator calls another.

There is a user defined in the Tomcat user file called

```
fedoraIntCallUser
```

with a fedoraRole of

```
fedoraInternalCall-1
```

(we shall come back to the Tomcat user file in a while). If you are going to build a security system where one disseminator may call another then this fedoraRole needs to be included in your permissions.

The second issue is that of IP permissions. The Fedora Security systems allows you to limit access to disseminators by IP address. It is therefore necessary that the IP restrictions on a disseminator that may be called internally allow the Fedora server to call itself - in other words, the server's own IP address must be allowed to call the disseminator.

Fedora have supplied a tool to manage IP addresses for backend security which we shall deal with later.

Apologies if these last two pages have seemed a little heavy going, but sorting out these issues before you start will potentially save you a lot of time and frustration. Now, where were we - ah yes, security...

Invoking Fedora security

When we started our journey I suggested that you would probably want to get to know Fedora with only limited security operating. I suggested that you use

```
fedora-setup no-ssl-authenticate-apim
```

so that the management interface was protected but that ordinary users did not face restrictions. This is all well and good but it is unlikely that it would be a satisfactory arrangement for a production system. We now need to turn on Fedora's security properly. Just before you rush into that, a question: You did change the fedora-base.fcfg file, didn't you?

When you change Fedora security levels, a new fedora.fcfg file is built from the base file. If you are not careful this could destroy all the changes you may have made to fedora.fcfg. I would suggest:

- copying the original fedora-base.fcfg file and keeping it somewhere safe, and
- incorporating into the fedora-base.fcfg file all the changes that you have made to fedora.fcfg.

This way, if you change security settings, you don't lose your basic configuration. The base file can be found in:

```
usr/local/fedora-2.1/server/fedora-internal-use/config
```

If you have sorted this out, you need to stop the server and run

```
fedora-setup no-ssl-authenticate-all
```

(drop the 'no-' if you are using ssl.) Then restart the server.

Probably the quickest way to check that security is now working is to try to use the search interface:

http://*yourBaseURL:port*/fedora/search

You should now be challenged for a username and password. The Fedora administrator entries should, of course, allow you access.

Directories and safe copies

Fedora's security system works by applying all applicable security scripts held below the directory

```
usr/local/fedora-2.1/data/fedora-xacml-policies
```

For the moment we are going to be working with policies in

```
usr/local/fedora-2.1/data/fedora-xacml-policies/repository-policies
```

In this directory is a sub-directory called 'default' which contains Fedora's basic group of scripts compatible with the security setting 'authenticate-all'. We strongly suggest you take up their suggestion of creating another directory at the same level for your own policies. We called ours simply 'hull'.

We also strongly suggest taking copies of these two directories somewhere outside the fedora-xacml-policies directory tree so that they can be put back in case of a problem.

Things have stopped working!

If all has gone well up to this stage you will be faced with the situation that a number of demonstration objects may have stopped working! Believe it or not, this is good. If you want to know what I'm talking about, search for 'demo:SmileyStuff' and have a look at its dissemination methods. You should find that 'list' works but that 'view' no longer does.

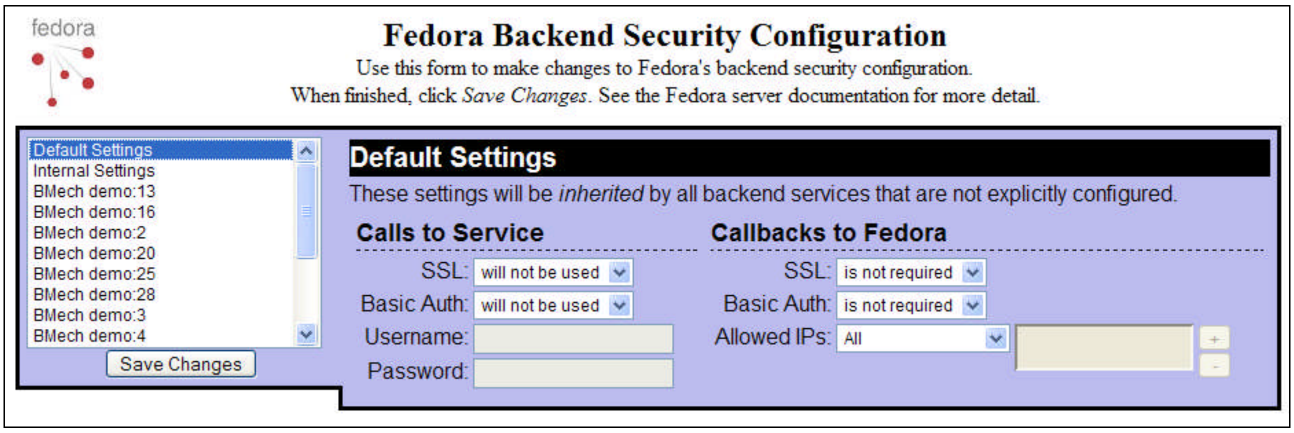
Earlier on in this section I described potential problems if a disseminator called another disseminator which was the subject of a security script. Here the BMech associated with 'demo:Collection' is calling the BMech associated with 'demo:DualResImage'. Under the demonstration security model provided by Fedora, this latter BMech has its IP access restricted to 127.0.0.1 (localhost). Likely you aren't using this IP address for your server any more.

Backend security management

Fedora provide a tool for backend security management at:

http://*yourBaseURL:port/* fedora/management/backendSecurity

(Be careful of the upper case 'S' in the last part of the URL.)



Fedora Backend Security Configuration

Use this form to make changes to Fedora's backend security configuration.
When finished, click *Save Changes*. See the Fedora server documentation for more detail.

Default Settings

These settings will be *inherited* by all backend services that are not explicitly configured.

Calls to Service

SSL: will not be used
Basic Auth: will not be used
Username:
Password:

Callbacks to Fedora

SSL: is not required
Basic Auth: is not required
Allowed IPs: All

If you scroll down the list of BMechs at the left, clicking on them as you go, you will find a number of them restricted to just 127.0.0.1, and a number restricted to other addresses.

I suggest that for each BMech, where the only allowed IP is 127.0.0.1, you add the IP of your own server and save the new settings. Leave those with other IP addresses as they are.

You also need to add your server's IP address to the 'Internal Settings' entry.

When you have done this, run the utility

```
fedora-reload-policies [http]
```

Remember the http can be omitted on Windows-based Fedora servers.

You should now find that demo:SmileyStuff's 'view' method works.

Now experiment with the demonstration material

Fedora provide a lot of demonstration scripts with the 2.1 download which you can now experiment with. They can be found at:

```
usr/local/fedora-2.1/server/userdocs/server/security/xacml-policies/examples
```

or similar. I would suggest copying the ones that interest you, one at a time, into your own security directory - either the one that you created like our 'hull' or another one you create, perhaps called 'demos'?

After you copy a policy remember to invoke:

```
fedora-reload-policies [http]
```

...and remove it before you try another policy. Be clear that these policies are not designed to work together in their totality; each is intended to demonstrate an aspect of security and to serve as a model should you wish to do something similar.

Once you have found policies that look to be the sort of thing you might need, you will want to try creating your own based on them. Remember to run

validate-policy *mypolicy*.xml

before testing them - and remember what I said about the utility OK-ing non-existent files.

This is a journey that you must undertake on your own. I found it quite straightforward once I had identified and solved some problems and had learnt a few tricks. I've given you the list, so for you it should be easier still.

Securing what?

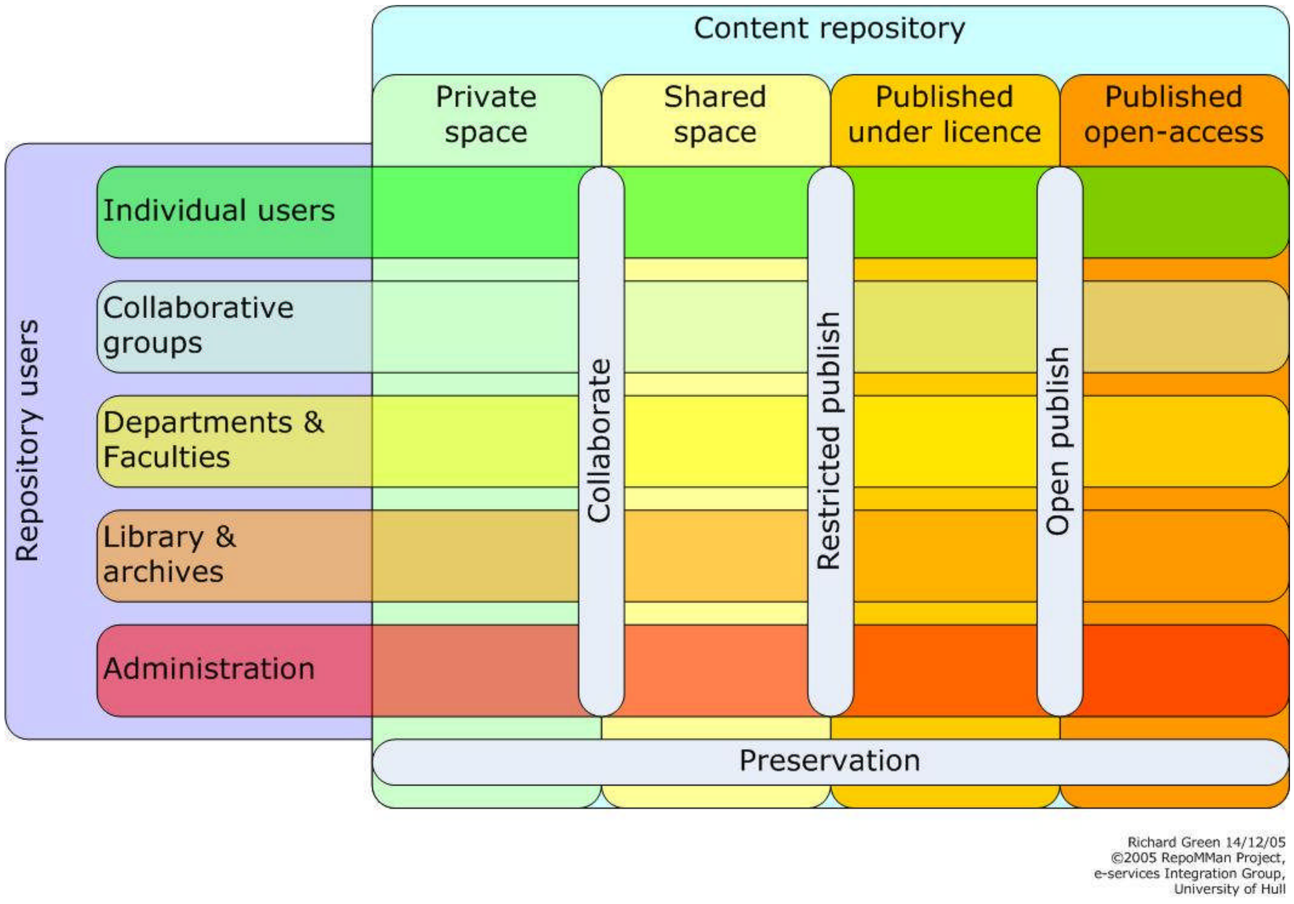
A much bigger pair of problems is 'What do you want to secure?' and 'How best to do it?' Again, I find it difficult to give you a quick answer, but perhaps I can give you some pointers based on our initial experience at Hull.

First, I'm afraid that you will need to understand a little bit about what we aim to do at Hull because our repository will not conform to the generally accepted model. This is challenging for us, because it will give us a whole raft of problems to solve, but perhaps good for you because your challenges are likely to be a subset of ours!

A security model for the University of Hull repository

Hull's view of a repository

Many institutions seem to view a digital repository in some sense as a home for 'finished products' - research output, theses, learning objects or whatever. At Hull we are developing our repository to be a working tool which will support the user throughout the creation of that product up to and including possible 'publication' in the repository. This can be represented diagrammatically as below:



Consider a researcher as an example of an individual user. (S)he has an idea and starts to develop it in the 'My Repository' area, a private space. Whilst it is there, the facilities of the repository ensure that the work-in-progress is backed up regularly and that it is available from anywhere that has an IP connection. At a later stage (s)he may wish to share it with others. It is moved into a 'shared space' where it is accessible to the researcher and a defined group of collaborators. The repository deals with record locking, so that conflicting revisions cannot occur, and versioning. Eventually a copy of the work may be put into one or other of the 'published' spaces where either a restricted group or anyone can have access to it, depending on licensing considerations etc. Throughout these stages, the researcher has the option to have the system help him or her to add metadata to the object so that it can be indexed and searched for. This may be useful even in a private space if it contains hundreds of files.

We want to provide a service, not only to researchers but also to all the other members of the university; staff (teaching and support), administrators, and potentially even to students.

Thus Hull's view of a repository is somewhat unusual by having 'pre-publication' content and potentially thousands of users (as opposed to visitors). In addition, we are aware that we shall have to be able to ingest a very wide range of file types.

The RepoMMan Project

Some of you will realise that this document is being produced by the JISC-funded RepoMMan Project at the University. The project and the development of the repository are not one and the same thing. The JISC funding is actually to develop a workflow tool which will enable users to interact with the repository in order to create repository objects, manipulate them, add metadata, and so on. Clearly, though, the project and the work on developing the repository itself are tightly interlinked.

Where to start: an image collection

The RepoMMan project started by analysing the needs of researchers within the institution. On the one hand this has enabled us to start developing a workflow tool and on the other has given us a starting point for the repository development. Many researchers need to manipulate collections of images, as well as texts, and we felt that an image collection would give us a useful, potentially visually attractive, demonstrator in the early stages of the project.

Accordingly, the University Photographic Service (UoHPS) provided us with a number of images and from them four(!) have been selected for the collection. (If it will work with four, it will likely work with 4,000 - the difference being a lot of unnecessary - at this stage - repetitive work.)

How to apply security? Well the photographs form part of a set that is regularly provided to staff who need 'general images of the University' to illustrate this or that. For the purposes of our R&D we postulated extending this group to students as well. We decided that 'screen-sized' and smaller images from the collection would be available to them, but that full-size originals, suitable for use in print media, would be available only to members of the University Photographic Service and to members of the University's Marketing & Communications (MarComms) group.

We have consulted quite widely about the possible structure of image objects for this work and have developed a 'standard colour image model' for them. This includes four image datastreams (archive, full-size, screen-size and thumbnail) and a metadata stream in two parts (essentially, descriptive metadata and technical image metadata). Our intention is that the archive datastream should be accessible only at administrator level, that the full-size image and technical metadata should be available only to UoHPS and MarComms, but that the screen-size and thumbnail images, together with the descriptive metadata, should be available to all staff and students. The images will not be available to the general public.

At the time of writing (2006-04-03) the image content model referred to in the last paragraph is still under development. That said we would be happy to share it with interested parties, though not yet via RepoMMan's website.

Securing the images

Fedora provides an almost bewildering list of 'hooks' on which security can be hung. At the highest level these could be 'all access to API-A' or similar, at the lowest level we could have a BDef state or a datastream MIME-type. The full list covers 'actions' such as getting a datastream, and objects, both at the object property and datastream property levels. In due course the list will also cover the 'user environment', such things as user IP address.

The full list can be found in a file called vocabulary.txt at:

usr/local/fedora-2.1/server/userdocs/server/security/xacml-policies/vocabulary.txt

Whatever security mechanisms are set up, Fedora evaluates all that apply directly or indirectly to the object and the action you are trying to perform on it. Only if all the applicable policies will allow you access do you ultimately get what you are seeking, any problem along the way and you get 'authorization denied".

Clearly, it is best to set security at the highest possible levels in the security tree so as not to end up with a bewildering array of policies.

This is our solution. Bear in mind that it is our first attempt and that we may live to regret it later.

All the collected images conform to the Fedora object model 'UoH_Std_Col_Img' and this fact is contained within the object's basic metadata. (If you are creating objects via the Fedora Admin Client, this is one of the first items you fill in for a new object: see page 9.) There is a security script that denies direct access to the datastreams for such objects to all but Fedora Administrators, allowing access only to UoHPS, MarComms, Staff and Students via an appropriate disseminator. This is a multi-resolution disseminator and it is the basis of a further security script which denies access to its getFullSize method to all but UoHPS and MarComms.

We have created two implicit collection objects in Fedora. One has member images that are available only to the UoHPS or MarComms groups, the other has images also available to all staff and students. At the moment the collections are identical, but this may not always be so. The collection objects have associated dissemination mechanisms; the first is the subject of a security script which limits it to UoHPS and MarComms, the second has a different security script limiting access to Staff and Students. The different disseminators each generate a webpage showing the Collection in a manner appropriate to their target users. The higher level users have links to access the FullSize image, Staff and Students can only link to the ScreenSize image.

You probably feel that you want to read that section over again, slowly, and perhaps with a cold compress on your head. I *think* it makes sense and gives us a fairly flexible basis on which to build other things. We have the basis for collections of images available to all staff and students, with higher resolution versions of the images available to a select few. We shall see how it works over time.

Fedora roles

This is all very clever, but just *how* are these user groups controlled in what they can do? Well the demonstration scripts should give you a good basic understanding of this and the group memberships are defined in the Tomcat user template at:

usr/local/fedora-2.1/server/jakarta-tomcat-5.0.28/conf/tomcat-users_fedoraTemplate.xml

For development work we have added a set of users as follows:

```
  <user name="student" password="abcde" roles="fedoraRole=student" />
  <user name="staff" password="fghij" roles="fedoraRole=staff" />
  <user name="UoHPS" password="klmno" roles="fedoraRole=UoHPS" />
  <user name="marComms" password="pqrst" roles="fedoraRole=marComms" />
```

The security scripts allow or deny access to users on the basis of their Fedora role. Of course, we have no intention of writing a Tomcat file for thousands of users. Eventually this information will come from the University's identity management system.

If you intend to create users in this way, remember that it is the template file that you should alter, not tomcat-users.xml. The latter is generated from the template each time that Fedora is restarted.

Privately owned objects

There is one class of object that we are developing which deserves special mention in this security section, and that is for objects which are privately owned.

The University's view of a repository is as a working tool, not just a place to deposit finished works. Accordingly we need to provide for objects that are works-in-progress; accessible only to the person developing them.

The solution that we are hoping for is to make all private objects members of a content model called 'user'. This is a pseudo-model in the sense that it will not impose any internal structure on the object, rather it is just a convenient label. When a 'user' object is called we shall have Fedora's security check the user's log-in ID against the object's owner. The Fedora documentation and some parts of the implementation (the search facility, for instance) reference an object property 'ownerId'. The folks at Fedora tell me that this is not implemented but is under serious discussion for inclusion in release 2.2 or 2.3.

Why am I telling you this? Because the security implementation caused us all sorts of grief and some of you may be interested in the solution which was developed with the help of a crucial insight from Ryan Scherle at Indiana University. What, in concept, was a very simple check was actually difficult to implement because of a lack of documentation. Don't get me wrong, this is not a dig at the Fedora team. The OASIS documentation for XACML doesn't cover this one!

As I said, the object property 'ownerId' is not yet implemented, so for the moment we are recording a private object's owner in the content model property and comparing this with the login ID. The rule goes like this:

```
<Rule RuleId="1" Effect="Deny">
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
      <!-- Compare object model with loginId (really needs to compare ownerId) -->
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
             AttributeId="urn:fedora:names:fedora:2.1:resource:object:contentModel" />
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
             AttributeId="urn:fedora:names:fedora:2.1:subject:loginId" />
        </Apply>
      </Apply>
    <!-- OR allow administrative access -->
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-
    of">
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            administrator</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            fedoraInternalCall-1</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
```

Deny access to the object unless the user's 'loginId' matches the object's 'contentModel' (this ultimately will be 'ownerId') or the user has administrator or internal rights.

Note the use of the 'string:one-and-only' function. This turns out to be the crucial bit of knowledge without which the script fails to deliver the required security. Some object properties can contain multiple values; the use of the one-and-only function guarantees to XACML that these are single-value-strings. (Thanks Ryan!!!)

References

Dublin core *The Dublin Core Metadata Initiative* http://dublincore.org/ (valid Jan 2006)
Fedora *The Fedora project* http://www.fedora.info/ (valid Jan 2006)
Oasis *eXtensible Access Control Markup Language (XACML) Version 2.0 - Committee draft 04* http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-04.pdf (valid Apr 2006)