Fedora Tutorial #2

Getting Started: Creating Fedora Objects and Using Disseminators

Version 10/25/2005 3:37 PM

Table of Contents

1 What is this document and who should read it? 4
2 What is Fedora and what does it do? 4
3 Why should you use Fedora? 5
4 How should you read this document? 6
5 Conventions used in this document 6
6 Getting Started: Using Fedora for Aggregating Content 7
6.1 Some basic definitions 7
6.2 Example 1: Making a document available in multiple formats 8
6.3 Example 2: Creating a surrogate for distributed content 13
7 Using Fedora to produce dynamic content 16
7.1 Example 3: Using built-in bdefs and bmechs 20
7.1.1 Ingesting the predefined bDef and bMech 20
7.1.2 Creating the digital object with appropriate datastreams 22
7.1.3 Creating the Disseminator 23
7.2 Example 4 - Modifying Example 3 using a redirect datastream 25
7.3 Example 5 – Writing your own bDefs and bMechs 26
7.3.1 The Web Service 27
7.3.2 Constructing the bDef 28
7.3.3 Constructing the bMech 30
7.3.4 Using your bDef and bMech to create a disseminator for an object 34
8 What's next? 35

Figures

Figure 1 - Fedora repository as mediator for services and content 5
Figure 2 – Fedora Administrator Login Screen 6
Figure 3 - New object dialog 8
Figure 4 - Configuring an object 9
Figure 5 - Datastream display 9
Figure 6 - Adding a new managed content datastream 11
Figure 7 - Complete datastreams for example 1 12
Figure 8 - Example 1 digital object and datastreams 13
Figure 9 - Adding a datastream with type Redirect 14
Figure 10 - Example 2 datastream display 15
Figure 11 - Example digital object and redirected datastream 16
Figure 12 - Layered view of Fedora context 17
Figure 13 - bMechs, bDefs, and disseminators in the Fedora context 19
Figure 14 – Dynamic dissemination access 20
Figure 15 – Example 3 disseminator creation 24
Figure 16 - Example 3 Dissemination access 24
Figure 17 - Dissemintor with redirect datastream 26
Figure 18 - bDef General Tab 28
Figure 19 - Abstract Method Definition 29

Figure 20 - Method Parameter Definitions 29
Figure 21 - bMech General Tab 31
Figure 22 - bMech Service Profile 31
Figure 23 - Method Properties window for greyScale 33
Figure 24 - Method Properties window for resize 34
Figure 25 - Image transformation disseminator 35

1 What is this document and who should read it?

This is an introduction for system developers and repository managers who are new to the Fedora open-source content management software. This is a hands-on tutorial. It assumes that you have already installed the Fedora software and are at a computer with access to a Fedora repository through the Fedora Administrator while reading this tutorial.

You don't have to have to be a programmer to understand and use this tutorial. However, you should be familiar with the operation and structure of web servers and web services.

This document is *not* intended for end users of content disseminated by a Fedora repository.

2 What is Fedora and what does it do?

Fedora is content management software that runs as a web service within an Apache Tomcat web server. Fedora provides the tools and interfaces for creation, ingest, management, and dissemination of content stored within a repository. There are a number of features that distinguish Fedora:

1. It supports the creation and management of digital content objects (from this point on called *digital objects*) that can aggregate data from multiple sources. For example, a digital object might be a set of `tiff` images that are the individual page images of a scanned document. The data sources may be either locally managed within the Fedora software or sourced from another URL accessible network server. The data sources may be content or metadata. You may think of these digital objects as advanced digital *documents,* especially in light of the feature described next.

2. It supports the association of web services with the digital objects. These services typically consume the data packaged within the digital object to produce dynamic disseminations from the digital object. For example, the digital object described above with multiple `tiff` page images may be associated with a service that OCRs the images that are components of the digital object and disseminates an `html` version of the pages. The services may be either local to the machine of the respective Fedora server or sourced another network accessible server that is addressable via a URL. In this manner, Fedora acts as a proxy layer that coordinates local and distributed data and web services within a uniform framework. This is illustrated in Figure 1.

3. It provides uniform access web-based interface to these digital objects, through REST requests and more powerful SOAP-based methods. This interface consists of a set of built-in methods to access characteristics common to all digital objects such as key metadata and internal structure. These include a method to introspect on an object to reveal the set of methods that constitute the extended behavior of that object. For example, a client could use these built-in methods to "learn"

about the capability of the digital object described above to dynamically disseminate an `html` page from a set of `tiff` images. The benefits of these are two-fold:

a. Clients accessing Fedora digital objects can rely on uniform access regardless of the nature of the object.

b. The disseminations available from an object are independent of the internal structure of the object. For example, the client interface of the example above in which `html` is disseminated from a set of source `tiff` pages could remain constant regardless of whether the underlying object contained `tiff` images, `jpeg`, `pdf`, or even simple static `html`. This gives the content developer great freedom to modify a repository's internals without disrupting the client and user views of the content.

4. It presents a uniform and powerful SOAP-based management interface. All internal operations of the repository such as object creation and management are available through this API, providing the hooks for integrating Fedora into a variety of environments. These makes Fedora useful as the foundation for advanced content management applications

5. It includes a comprehensive versioning framework that tracks the evolution of objects and provides access to earlier versions.

6. It includes a basic relationship framework for representing the links among digital objects.

7. It supports ingest and export of digital objects in a variety of XML formats. This enables interchange between Fedora and other XML-based applications and facilitates archiving tasks.

A number of these features are illustrated in Figure 1.

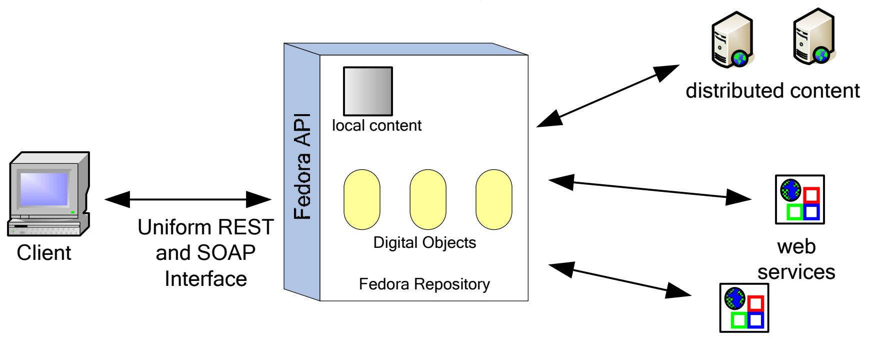


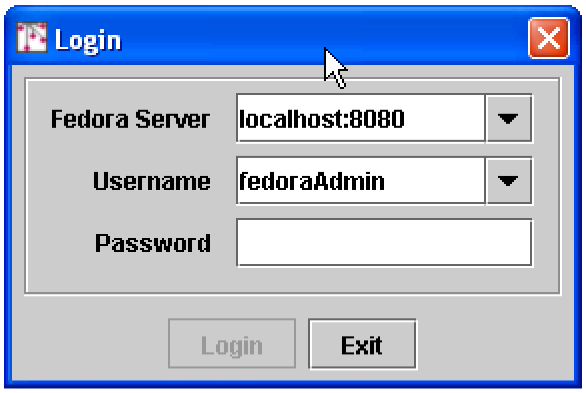
Figure 1 - Fedora repository as mediator for services and content

3 Why should you use Fedora?

Fedora may be the wrong choice for management of simple static web pages. There are a number of excellent tools for `html` editing and web site creation. Fedora is more appropriate for more advanced content management tasks. These include management of content and associated metadata, multiple versions of content, content available in multiple formats, and dynamically generated content from local and dynamic sources.

4 How should you read this document?

This document is intended to be hands-on, with you trying the examples on a running Fedora repository. You should therefore, have already downloaded and installed Fedora, and started a server. You should then access the Fedora repository by running the Fedora Administrator interface, `fedora-admin`, which is located in the `FEDORA_HOME/client` directory (you can start this program from the command line if you have configured your environment variables properly). Upon starting up the administrator interface you will be presented with the login screen shown in Figure 2. This document assumes that you have not changed any of the configuration defaults for your Fedora server so the Password you enter should be **fedoraAdmin**. If you have changed your configuration values or are running the Feodra Administrator from a machine you will need to change the values in the Login screen appropriately.



Fedora Server: localhost:8080
Username: fedoraAdmin
Password:

Figure 2 – Fedora Administrator Login Screen

You should read this document in order, since later examples assume knowledge of techniques and definitions introduced earlier.

5 Conventions used in this document

The font conventions used are:

- *Defined terms are introduced like this.*
- **Text in dialog boxes and windows is shown like this.**
- `URLs, directory paths, file names, and similar items are shown like this.`

All pathnames assume that you have set your `FEDORA_HOME` environment variable and descend from the directory defined by that variable.

All URLs that access the Fedora repository assume that the host:port of the repository is `localhost:8080`.

6 Getting Started: Using Fedora for Aggregating Content

This section describes how to create digital objects in Fedora that aggregate data from multiple sources. The examples demonstrate how to do this with both local data and data from networked sources. This section provides the foundation for the next section, which describes how to use Fedora to create dynamic content by exploiting web services. Make sure you understand the basic concepts here, before moving on to that next section

6.1 Some basic definitions

To understand content aggregation in Fedora, you need to be comfortable with two terms:

1. *Digital Object* – This is the basic unit for information aggregation in Fedora. At a minimum a digital object has:

    a. An identifier or PID. This PID provides the key by which the digital object is accessed from the repository.

    b. Dublin Core metadata that provides a basic description of the digital object.

2. *Datastream* – A component of a digital object that represents a data source. A digital object may have just the basic Dublin Core datastream, or any number of additional datastreams. Each datastream can be any mime-typed data or metadata, and can either be content managed locally in the Fedora repository or by some external data source (and referenced by a URL). When you create a new datastream in a digital object, you assign it to one of four types, or *control groups*, depending on the nature of the data that it represents.

    a. *Managed Content (M)*: Datastream content is stored and managed within the Fedora repository's persistent storage. The content can be any MIME type including XML.

    b. *Inline XML (X)*: A special case of M, restricted to well-formed XML. In this case the datastream content is stored as part of the XML structure of the digital object itself and is thus included when the digital object is exported (e.g., for archival purposes).

    c. *Externally Referenced (E):* Datastream content is external to the Fedora repository and is referenced by a URL that is recorded within the digital object. The content can be any MIME type including XML.

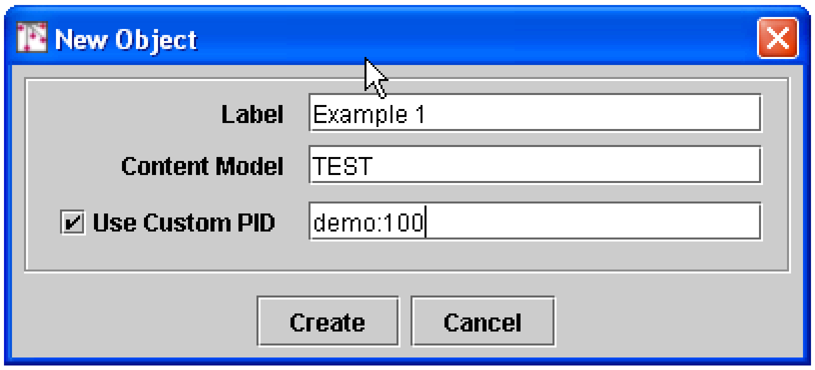
    d. *Redirected Content (R):* Like E, but datastream content is delivered to the client without any mediation by Fedora; i.e., via an HTTP redirect. You should use this datastream type when the external content is a web page with relative links or it is streaming audio or video. The content can be any MIME type including XML.

Decisions about what to include in a digital object and how to configure its datastreams are basic modeling choices as you develop your repository. The examples in this tutorial demonstrate some common models that you may find useful as you develop your application.

6.2 Example 1: Making a document available in multiple formats

It is often useful to provide access to a digital document in several formats. For example an ePrints server might provide `html` for those who wish to render the document in a browser, `pdf` for those who wish to view the document with author-determined formatting, and `TeX` for those who wish to access and use the document source. This example demonstrates how to construct a digital object where each datastream corresponds to an available format. More advanced techniques, demonstrated later in this tutorial, make it possible to achieve the same results by generating formats dynamically from a single base format. But for now, we'll stick to simple static aggregation.

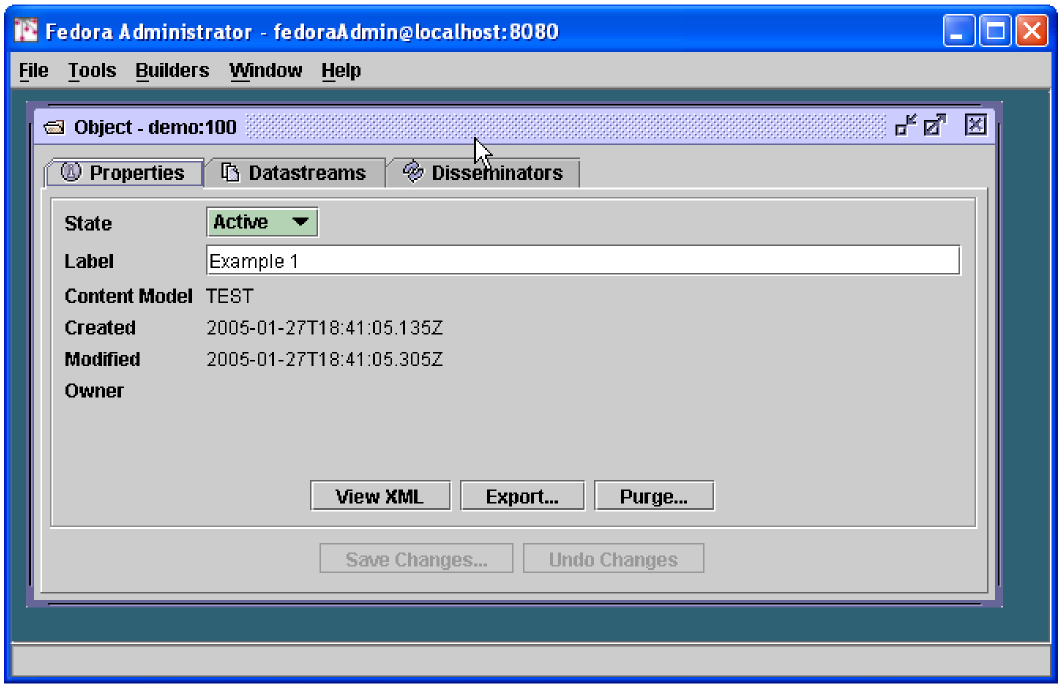
Start by selecting **File/New/Data Object** in the Admin GUI. Complete the **New Object** dialog box as shown in Figure 3.



Label: Example 1
Content Model: TEST
☑ Use Custom PID: demo:100

Figure 3 - New object dialog

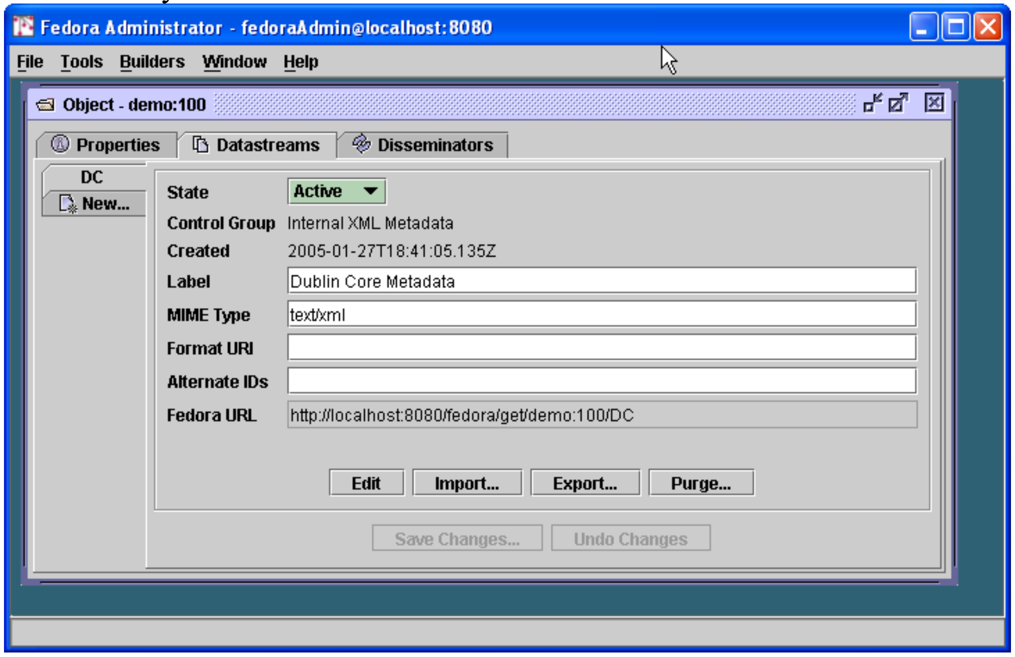
The **Content Model** field is available for more advanced applications and can be left blank for now. Also, check the box for **Use Custom PID** and enter **demo:100**. Note that when you do not assign your own PID, the Fedora repository will create one for you. Select the **Create** button and you should see a window like that in Figure 4. Observe that the PID of the created object (in this case `demo:100) is` displayed in the title bar.



State: Active
Label: Example 1
Content Model: TEST
Created: 2005-01-27T18:41:05.135Z
Modified: 2005-01-27T18:41:05.305Z
Owner:

Figure 4 - Configuring an object

Since our task here is to define the datastreams in the object, click on the **Datastreams** tab and you will see a window like that in Figure 5. Note that at this point there is only one datastream in the object – the DC datastream for basic descriptive metadata that was automatically created by Fedora. You can select that datastream and select the **Edit** button to see the default contents of this Datastream, with the DC `title` and `identifier` fields already filled in.



State: Active
Control Group: Internal XML Metadata
Created: 2005-01-27T18:41:05.135Z
Label: Dublin Core Metadata
MIME Type: text/xml
Format URI:
Alternate IDs:
Fedora URL: http://localhost:8080/fedora/get/demo:100/DC

Figure 5 - Datastream display

A few points to note about what you have done so far:

- You will notice that the **Control Group** of the DC datastream is **Internal XML Metadata**. As explained earlier, Fedora has a number of control group types, of which this is one. This type is appropriate for metadata that is represented in XML – Dublin core metadata being one example. A digital object can have multiple metadata datastreams, for example MARC, LOM, Dublin Core, and others.

- You can directly edit the Dublin Core metadata – e.g., add new Dublin Core fields - by selecting the **Edit** button and modifying the contents of the text pane. . When you press **Save**, Fedora will check that the datastream is well-formed XML.

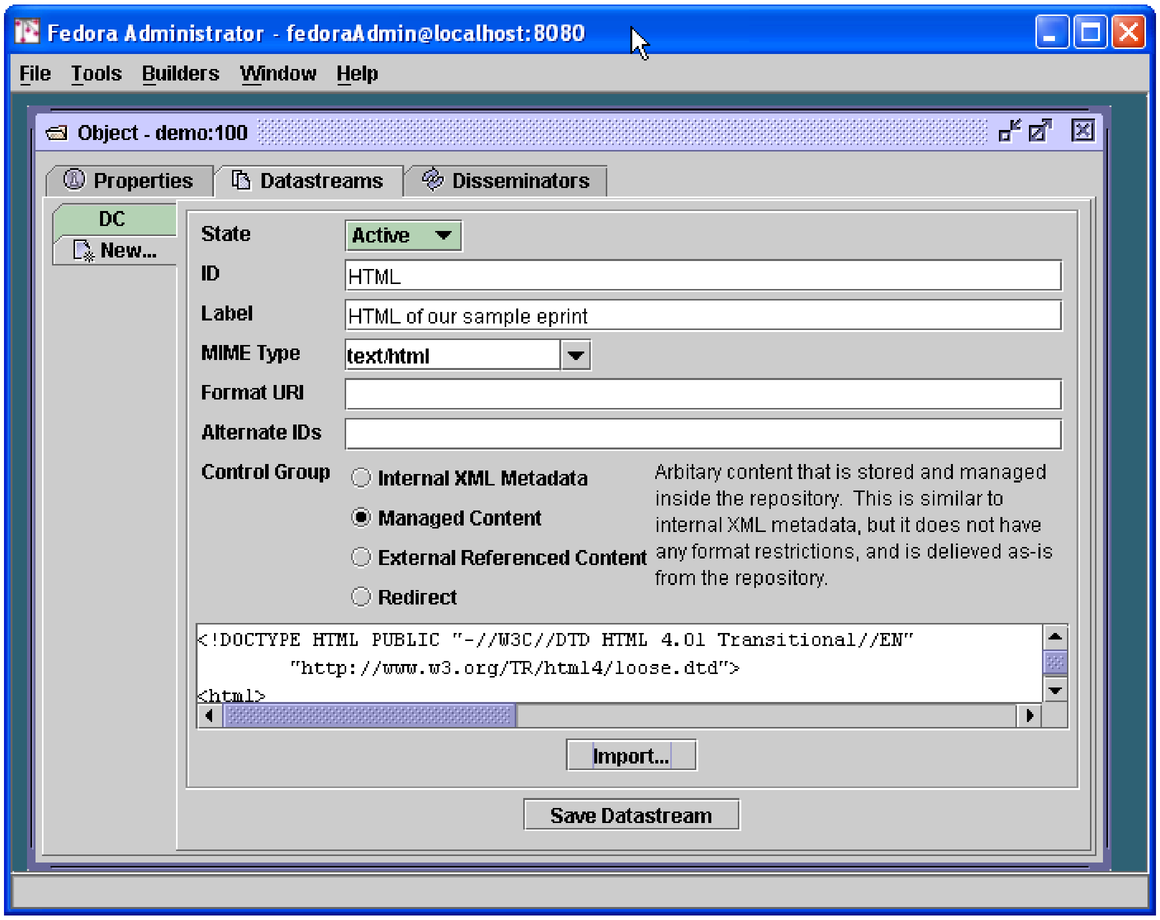
You may also create Dublin Core metadata (or any other XML-based metadata) in an external XML editor and use the **Import** button to replace the datastream with this data. When you press **Save**, Fedora will check that the datastream is well-formed XML.

You will notice that there are optional fields on the datastreams pane for **Format URI** (to refine the media type meaning with a URI that identifies the media type) and **Alternate Ids** to capture any other existing identifiers you would like to associate with a datastream. We will not be using these in this tutorial.

It is now time to add the eprint document formats as new datastreams. You can find content for creating the datastreams in this example in:

```
FEDORA_HOME/userdocs/tutorials/2/example1/artex.html
FEDORA_HOME/userdocs/tutorials/2/example1/artex.pdf
FEDORA_HOME/userdocs/tutorials/2/example1/artex.tex
```

To do this, select the **New…** tab on the left side of the window. We'll start with the **html** format. To insert data into the datastream, you use the **Import…** button. This presents a dialog that will allow you to import from your local file system or from a URL.
Your completed HTML datastream should look like the dialog as shown in Figure 6 (after you have imported the content).



State: Active
ID: HTML
Label: HTML of our sample eprint
MIME Type: text/html
Format URI:
Alternate IDs:
Control Group: ( ) Internal XML Metadata [✓ Managed Content] ( ) External Referenced Content ( ) Redirect

Arbitary content that is stored and managed inside the repository. This is similar to internal XML metadata, but it does not have any format restrictions, and is delieved as-is from the repository.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

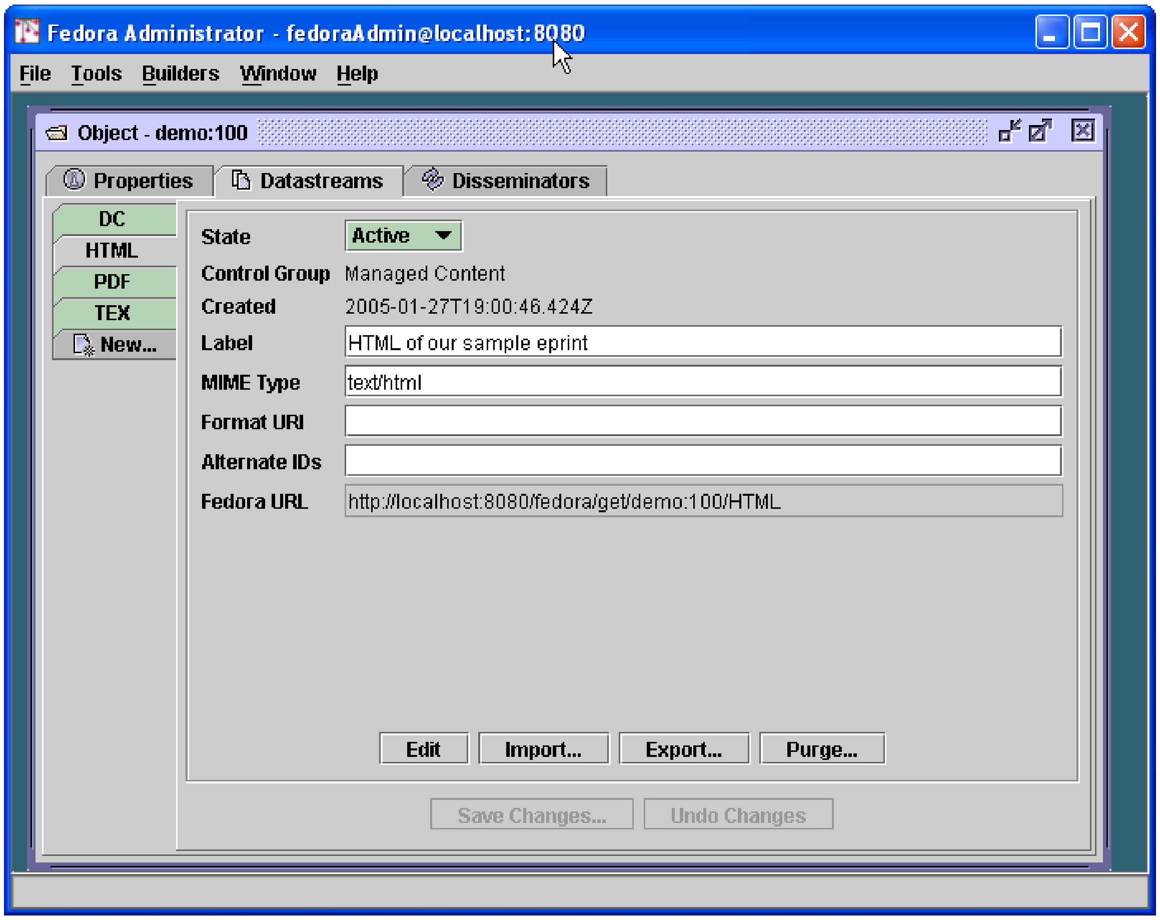
Figure 6 - Adding a new managed content datastream

A few notes on the contents of this dialog:

- The **ID** of the datastream should be a single token. By convention, it describes the purpose of the datastream.
- The **Label** can be a longer, more descriptive string.
- Note that the **Control Group** is **Managed Content**. As shown in the descriptive text this datastream type is appropriate for any type of data (mime type), in contrast to **Internal XML Metadata**. Once you select this radio button, you can select from the variety of **Mime Types** of the managed content – in this case `text/html`.

You can now select the **Save Datastream** button and repeat the same process to add the `pdf` and `Tex` datastreams. For the `pdf`, you can select **Mime Type: application/pdf** and import the file `ex1.pdf`. For `Tex`, you can select **Mime Type: text/plain** and import the file `ex1.tex`. In each case you should enter appropriate **IDs** and **Labels**.

You're done! Your **Datastreams** window should now look something like that shown in Figure 7, showing all the datastreams you have entered in the left-side tabs in the window.



State: Active
Control Group: Managed Content
Created: 2005-01-27T19:00:46.424Z
Label: HTML of our sample eprint
MIME Type: text/html
Format URI:
Alternate IDs:
Fedora URL: http://localhost:8080/fedora/get/demo:100/HTML

Figure 7 - Complete datastreams for example 1

You will notice as you click through each datastream that there is a **Fedora URL**, giving the unique URL to access each datastream from the Fedora repository. Try going to a browser and entering one of these URLs – the browser will download the datastream and display it. These URLs can be used by web applications and REST-based web services that access datastreams from Fedora digital objects. Note that if you are building SOAP-based web services, there are also SOAP methods (`getDatastream` and `getDissemination`) that provide digital object access. You can also try entering the root URL for the entire digital object, which is simply the common prefix of all the datastream URLs – e.g., `http://localhost:8080/fedora/get/demo:100`. This accesses the header page for the digital object, which allows you to access its datastreams (available through the **item index** hyperlink) and disseminations (available through the **dissemination index** hyperlink).

Figure 8 illustrates the structure of the object you have created and the correspondence of REST-based access requests to the object and its components (via API-A-LITE).

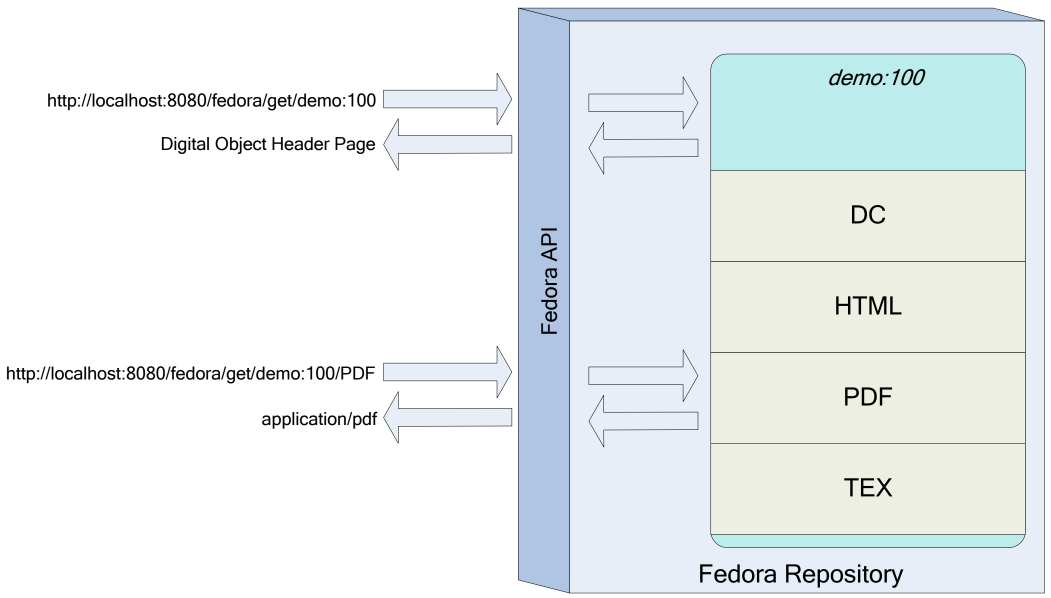

http://localhost:8080/fedora/get/demo:100
Digital Object Header Page
http://localhost:8080/fedora/get/demo:100/PDF
application/pdf
Fedora API
demo:100
DC
HTML
PDF
TEX
Fedora Repository

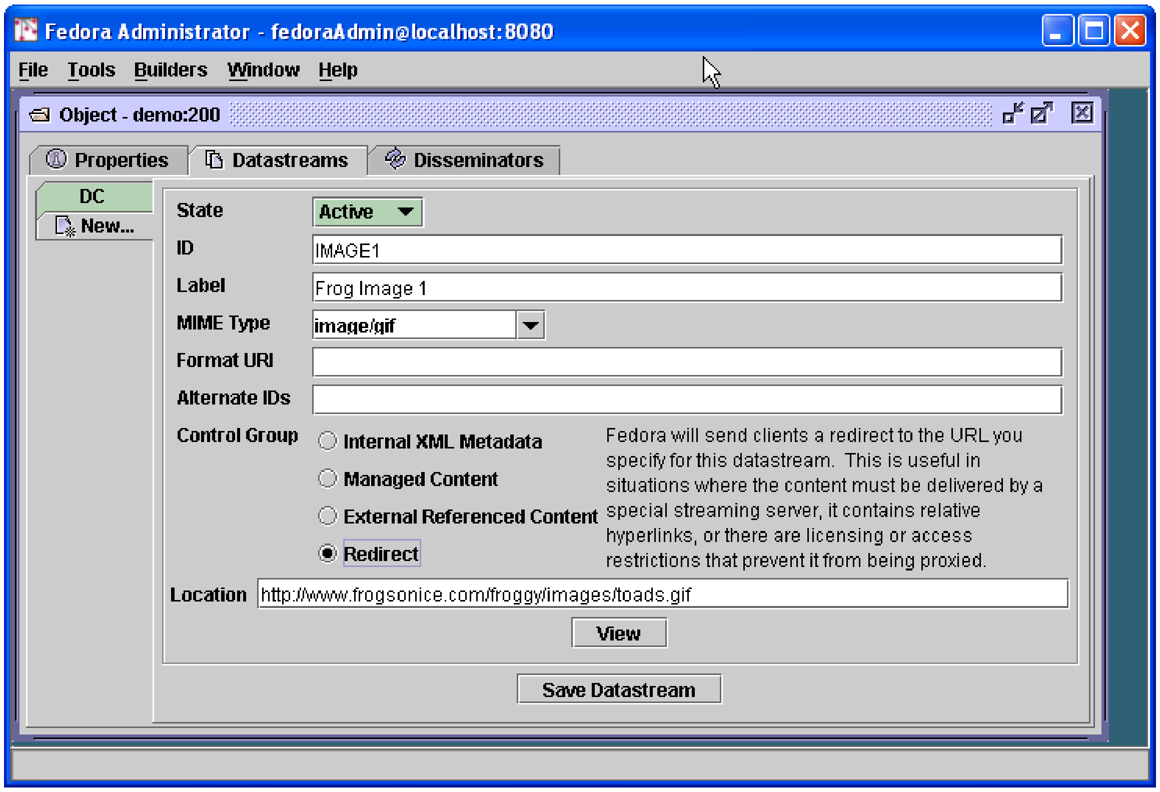
Figure 8 - Example 1 digital object and datastreams

6.3 Example 2: Creating a surrogate for distributed content

The previous example demonstrated how to aggregate imported content into a Fedora digital object. There are many reasons why importing content into a repository might not be appropriate such as rights restrictions or the dynamic nature of the content. To accommodate these restrictions, digital objects in Fedora may contain datastreams that reference externally managed content, and in fact may mix local and distributed data sources.

This section describes how to do this where the motivating example is the creation of a hypothetical learning object in an educational digital library, such as the NSDL. The digital object created in this example combines three frog images from the NSDL collection and some locally-managed text.

To get started follow the same procedure as illustrated in Figure 3, this time entering **Example 2** as the Label and **demo:200** as the custom PID. As in Example 1, select the **Datastreams** tab and then enter the information as shown in Figure 9.


Fedora Administrator - fedoraAdmin@localhost:8080
File Tools Builders Window Help
Object - demo:200
Properties
Datastreams
Disseminators
DC
New...

State: Active
ID: IMAGE1
Label: Frog Image 1
MIME Type: image/gif
Format URI:
Alternate IDs:
Control Group: Internal XML Metadata [ ] Managed Content [ ] External Referenced Content [ ] [✓ Redirect]

Fedora will send clients a redirect to the URL you specify for this datastream. This is useful in situations where the content must be delivered by a special streaming server, it contains relative hyperlinks, or there are licensing or access restrictions that prevent it from being proxied.

Location: http://www.frogsonice.com/froggy/images/toads.gif

Figure 9 - Adding a datastream with type Redirect

You will enter the datastream identifier of **IMAGE1**, a label for this datastream, and then information about the content. The content is of MIME type **image/gif**. You should select the Control Group of **Redirect,** and the enter a URL that specifies the Location of the image file, specifically:

```
http://www.frogsonice.com/froggy/images/toads.gif
```

A few notes on the contents of this dialog:

- Pertaining to the selection of a Control Group, you have two choices if you want the datastream to point to content that resides outside the Fedora repository (**External Referenced Content** *and* **Redirect).** In this case we chose **Redirect** . To review, the meaning of the two options for mapping to external content are:
    - **External Referenced Content** is useful when you want Fedora to mediate access to the datastream, for example when you want to hide the source URL from the user. Fedora mediates access to these datastreams, meaning that the content is streamed through the Fedora server.
    - **Redirect**. makes use of a simple HTTP redirect to provide the content. This is useful when there are relative hyperlinks in the external content, but reveals the source URL to the user.

- Make sure that the **MIME Type** choice matches that of the content offered by the external source, in this case `image/gif`.

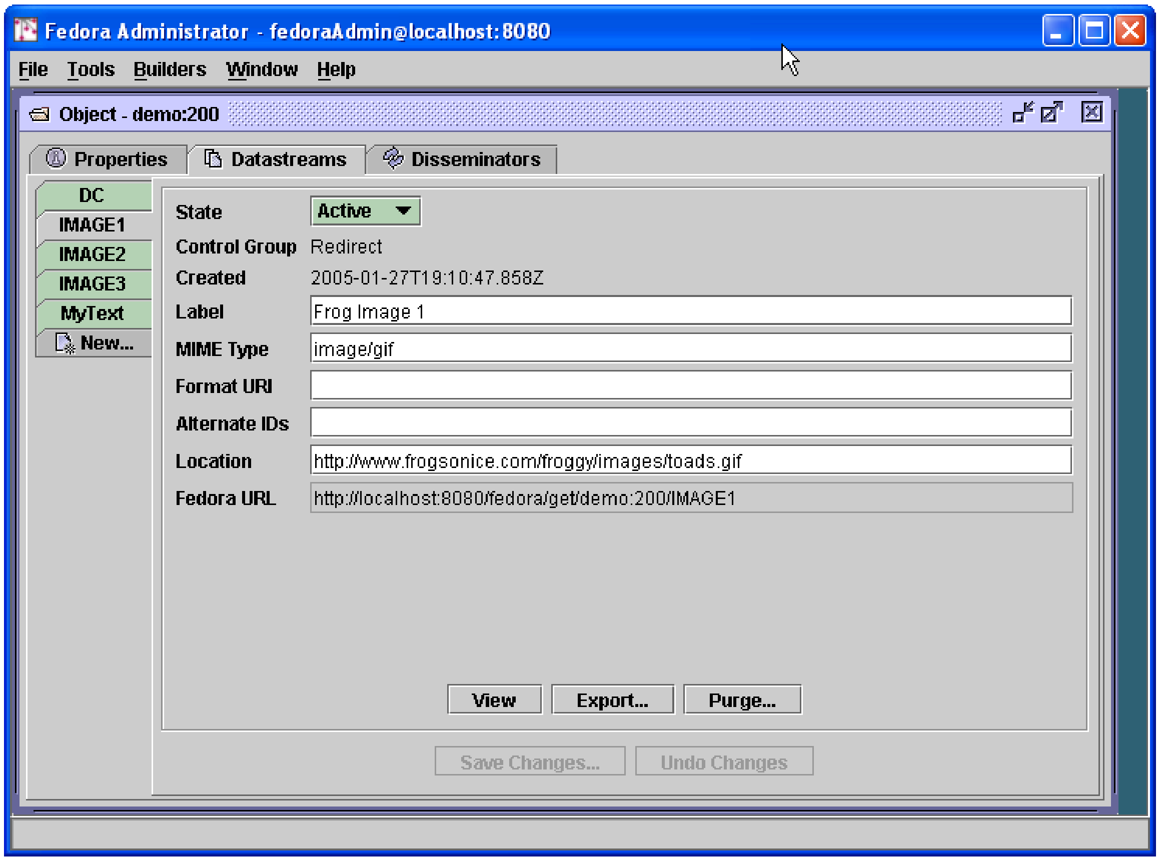
In the same manner, you can now proceed to add the two other datastreams with locations:
`http://www.werc.usgs.gov/fieldguide/images/hycafr.jpg` and
`http://www.nbii.gov/disciplines/herps/images/chira_leopard.jpg`

You should respectively identify these datastreams as **IMAGE2** and **IMAGE3**. (Note that if these sample URLs are no longer active, you can enter other URLs pointing to jpeg images to complete this tutorial exercise.)

Finally, add another datastream labeled **MyText** (containing some descriptive text about the images), with MIME Type `text/html`. Assign this datastream a Control Group of **Managed Content** indicating that the content will be imported and stored permanently in the Fedora repository. Import the content from the following location:

`FEDORA_HOME/userdocs/tutorials/2/example2/ex2.html.`

The resulting datastream window should now look like that shown in Figure 10.



State	Active
Control Group	Redirect
Created	2005-01-27T19:10:47.858Z
Label	Frog Image 1
MIME Type	image/gif
Format URI	
Alternate IDs	
Location	http://www.frogsonice.com/froggy/images/toads.gif
Fedora URL	http://localhost:8080/fedora/get/demo:200/IMAGE1

Figure 10 - Example 2 datastream display

Your done! Figure 11 illustrates the role of the redirected datastream at the time of digital object access via the Fedora REST-based interface (API-A-LITE). You can see this by going to the digital object profile page at:
`http://localhost:8080/fedora/get/demo:200`

You can access the datastreams for this digital object by viewing the item linked to from the object profile page. Then, select the link for one of the redirected datastreams. Fedora will redirect your browser to the location of the datastream content, without streaming the content through the Fedora repository server.

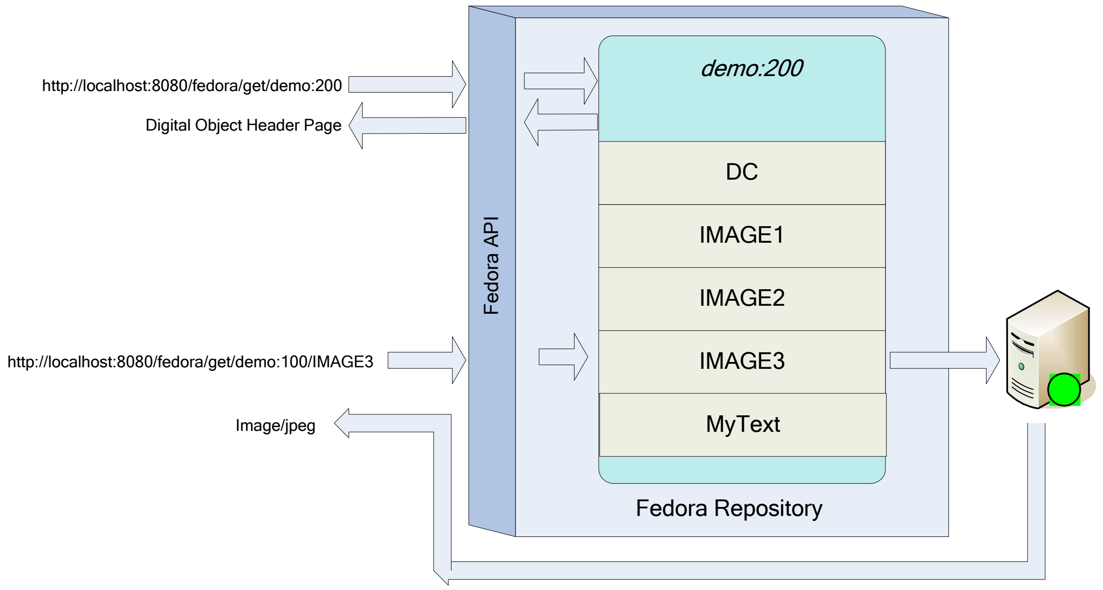

http://localhost:8080/fedora/get/demo:200
Digital Object Header Page
http://localhost:8080/fedora/get/demo:100/IMAGE3
Image/jpeg
Fedora API
demo:200
DC
IMAGE1
IMAGE2
IMAGE3
MyText
Fedora Repository

Figure 11 - Example digital object and redirected datastream

7 Using Fedora to produce dynamic content

The examples described so far demonstrate the basic content aggregation features of Fedora. As mentioned already, the power of Fedora lies in its ability to associate the data in a digital object with web services to produce dynamic disseminations. Some examples of this capability are as follows:

- Rather than packaging multiple formats of a document as in Example 1, it is possible to have a digital object with one datastream in a source format (e.g. `Tex`) and then associate a service with the digital object to transform the source format into multiple output formats (e.g. `pdf` and `html`). An obvious advantage of this is that any changes to the source format propagate out to the derived formats. Furthermore, less content is stored and/or duplicated in the repository.

- Rather than packing multiple metadata XML-based metadata formats in a digital object, it is possible to package a single base metadata format in a digital object (for example, fully qualified Dublin Core) and use that base format as the basis of metadata crosswalks. To do this, one could associate an XSLT engine (e.g. saxon) service with the digital object that processes the base format with a transform

XSL document (packaged as a datastream in another digital object) to derive one or more additional formats.

In both cases, static and dynamic, disseminations are available via REST or SOAP requests from clients to the Fedora Access service (API-A and API-A-LITE). The nature of the disseminated content – the format of the underlying data, where it is located, and whether it is static or dynamically generated – is invisible from the client perspective. As a result, a repository manager can significantly alter the nature of a digital object and the web services that it uses while maintaining the same interface vis-à-vis the client. Correspondingly, two digital objects with entirely different structure can appear the "same" from the perspective of consuming clients.

The remainder of this section presents a series of examples demonstrating how to create digital objects that exploit web services. The initial examples make use of services available in the Fedora software release (they run as "local services" within the Fedora server container). Later examples demonstrate how to construct your own custom objects with external web services. Before proceeding with the examples, this introduction summarizes the concepts and defines the terms used in the examples. Don't worry if the concepts are not entirely clear at first. You should read them now and then refer back to them as you work through the examples.

Figure 12 shows Fedora in the context of a number of related layers.

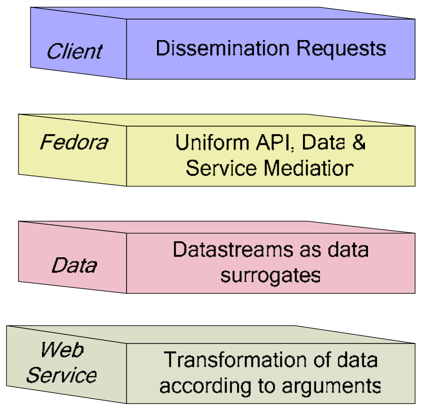

Client
Dissemination Requests
Fedora
Uniform API, Data & Service Mediatior
Data
Datastreams as data surrogates
Web Service
Transformation of data according to arguments

Figure 12 - Layered view of Fedora context

These layers are:

1. *Client:* Clients access content disseminations from Fedora through the Fedora Access service API (i.e., API-A-LITE and API-A). The uniform interface includes operations for discovering and accessing augmented behaviors of object. The mechanism for augmenting object behavior is described in this section.

2. *Fedora Repository Service:* Fedora maps client requests on the API into required data inputs and web service invocations to produce disseminations.

3. *Data:* Datastreams form the basis of disseminations. As shown in the previous examples, datastreams can be directly accessed via the Fedora Access service. Also, can serve as input to another "virtual" dissemination that is produced when the Fedora repository service calls upon another web service at runtime.

4. *Web Services:* These are web-accessible programs that are invoked by HTTP. In this tutorial we will describe how Fedora interacts with simple REST-based services where service operations are invoked using simple URLs (which contain arguments) and produce output via HTTP responses. We will not discuss Fedora interacting with other SOAP-bases web services here.

The process of creating digital objects with dynamic content disseminations involves creating linkages between these layers. During this process you will create and employ the following:

- *Behavior Definition (*or *bDef*): A digital object that is a template for client-side functionality, defining a set of abstract operations (methods) and their client-side arguments. Association of a bDef with a digital object augments the basic behavior of the object with the operations defined in the bDef template. A bDef may be associated with more than one digital object, thereby augmenting all of them with the same operations.

- *Behavior Mechanism (*or *bMech)*: A digital object that registers within Fedora the capability of web service(s) to perform the operations defined by a specific bDef. This registration includes defining service binding metadata encoded in the Web Service Description Language (WSDL) and also a *data profile* of the bMech. The data profile defines the types of inputs that are considered compatible with the service. In particular it declares the MIME types that are needed by the respective web service to perform its task. Multiple bMechs may be registered for an individual bDef, thereby exposing a generic client-side interface (defined by the bDef) over multiple data and web service foundations (defined by the bMechs).

These two kinds of special Fedora objects are stored in Fedora repositories. The set of all bDefs represents a "registry" of all the kinds of abstract services supported by the Fedora repository. The set of all bMechs represents a "registry" of all the concrete service bindings for the abstract service definitions supported by the Fedora repository.

At the end of the day, other digital objects make references to bDefs and bMechs as the way of providing extended access points for digital objects (i.e., dynamic content disseminations). This is done by adding a special component to a digital object known as a *Disseminator:*

- *Disseminator:* A component added to any digital object that ties together a bDef and a bMech in the context of that digital object to produce extended service-based functionality for that digital object. This binding involves the following:

- From the client perspective, it enhances the set of access points that are available on the digital object, beyond just the direct datastream access points. The disseminator associates the service operations defined by the bDef with the digital object. Multiple disseminators may be added to an individual digital object thus augmenting it, from the client perspective, with the operations of multiple bDefs.

- It establishes the linkage between the data profile of the bMech and specific datastreams in the object. These datastreams will play the role of "input" to the service defined by the bMech. The datastream MIME types must match the MIME types specified in the bMech data profile.

- It defines how an invocation of an operation in the associated bDef is translated to a URL that invokes the web services registered by the bMech.

Figure 13 refines the information in Figure 12 with the definitions described above. For the sake of illustrating relationships, the "Disseminator" is shown as a component outside the "Fedora" layer. In fact, it is integral to Fedora, as indicated by its shared color with the fedora layer.

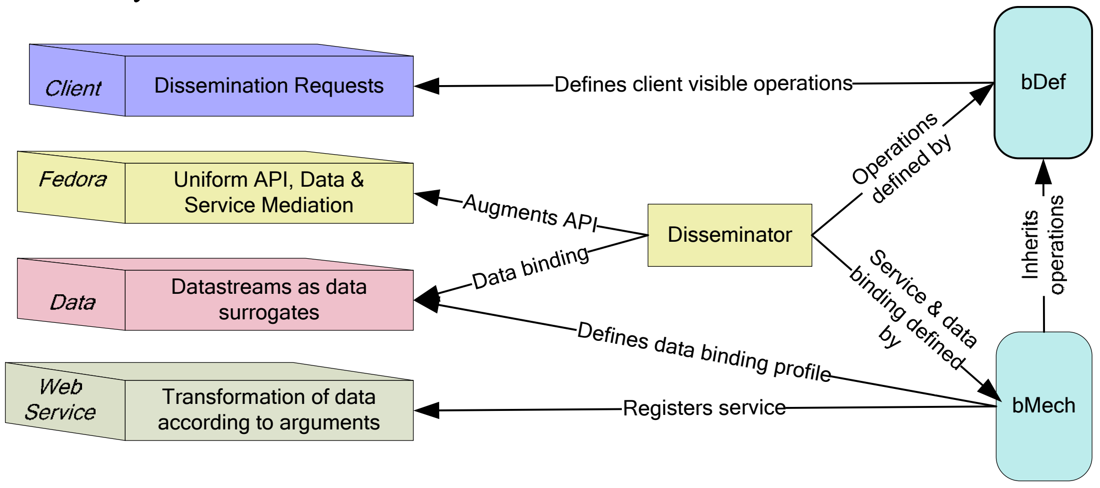
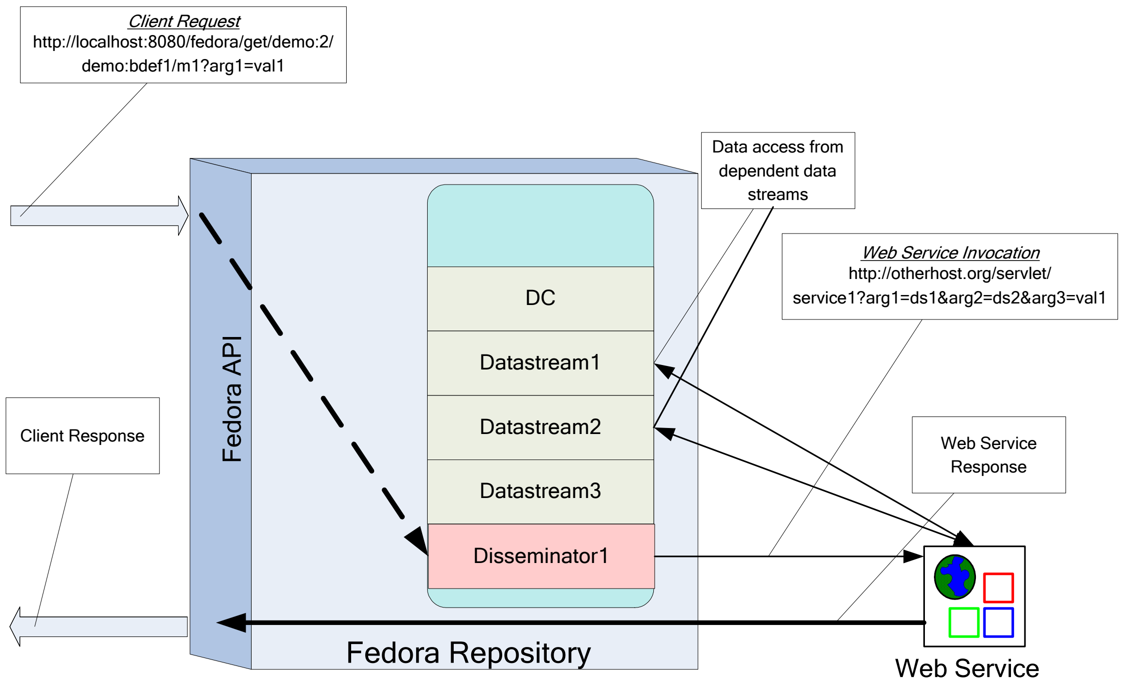

Client
Dissemination Requests
Fedora
Uniform API, Data & Service Mediation
Data
Datastreams as data surrogates
Web Service
Transformation of data according to arguments
Defines client visible operations
Augments API
Data binding
Disseminator
Operations defined by
Service & data binding defined by
Defines data binding profile
Registers service
Inherits operations
bDef
bMech

Figure 13 - bMechs, bDefs, and disseminators in the Fedora context

Figure 14 illustrates the interactions among Fedora, digital object datastreams, and web services in response to a dissemination request. As indicated, a client makes a request to the Fedora API (with a URL in this case), the disseminator invokes the respective web service via a URL, the web service fetches data from datastreams that are bound to the disseminator, and passes a response back to the client through Fedora.



Client Request
http://localhost:8080/fedora/get/demo:2/demo:bdef1/m1?arg1=val1
Data access from dependent data streams
Web Service Invocation
http://otherhost.org/servlet/service1?arg1=ds1&arg2=ds2&arg3=val1
Fedora API
DC
Datastream1
Datastream2
Datastream3
Disseminator1
Client Response
Web Service Response
Fedora Repository
Web Service

Figure 14 – Dynamic dissemination access

7.1 Example 3: Using bDefs and bMechs

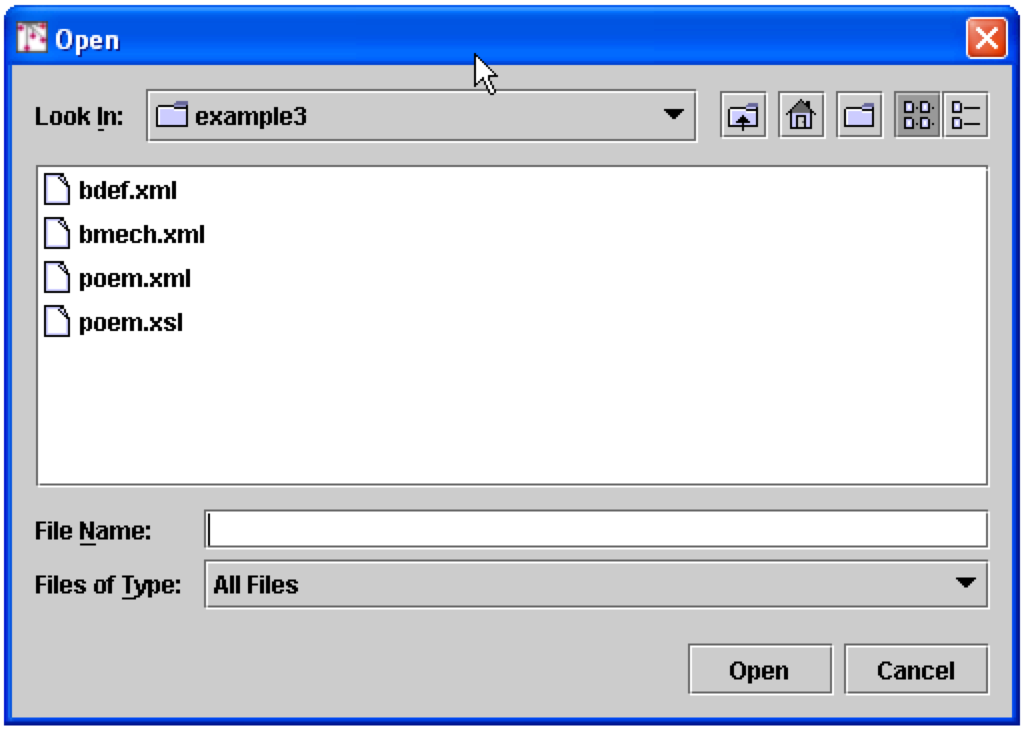
This example makes use of a bDef and bMech supplied with the Fedora tutorial. This will help you understand the basics dynamic disseminations in Fedora without writing a bDef or bMech. The next example describes how to do that more advanced task.

The web service used in the example performs an XSLT transform using the well-known saxon XSLT processor. This service requires two inputs, an XML source document and a XSL transform document. In this example, both of these XML documents are stored as managed content in a Fedora digital object. The XML source is data for a poem with tags for the structural elements of the poem (stanzas and lines). The XSL transform produces a HTML output of the poem that can be viewed in a browser. This example is borrowed from the web available source for Michael Kay's excellent XSLT book.

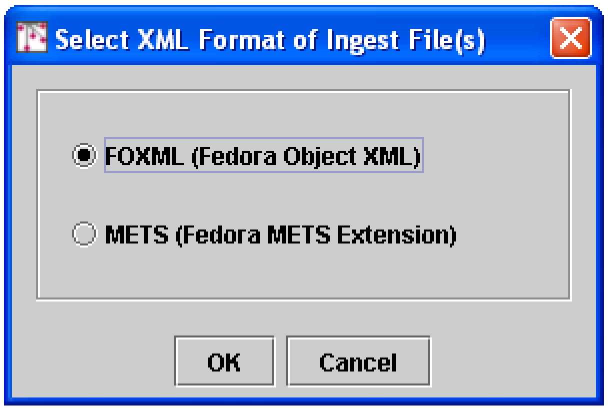
7.1.1 Ingesting pre-defined bDef and bMech objects

First we will ingest a sample bDef object into the repository.

Select **File/Ingest/One Object/From File…** in the Fedora Administrator. This will bring up a file selection dialog box as follows:



Browse the file system to select the ingest file for the bDef object whose file name is `FEDORA_HOME/userdocs/tutorials/2/example3/bDef.xml`. Since this ingest file is encoded as FOXML select the FOXML radio button as below:



[✓ FOXML (Fedora Object XML)] METS (Fedora METS Extension)

This will create the digital object with PID **`demo:ex3bDef`** in your repository. This bDef defines one method `getContent`. This generic method name is intentional – one could imagine this one bDef being used as the basis for several bMechs, each of which produces "content" via a unique transformation of an underlying source. This is one of the advantages of Fedora – providing a common interface despite multiple underlying representations.

Follow the same procedure to ingest a sample bMech object into the repository. Select the file `FEDORA_HOME/userdocs/tutorials/2/example3/bMech.xml`. This will create the digital object with the PID **demo:ex3bMech.** This bMech represents a concrete implementation of the abstract service operations defined in the bDef `demo:ex3bDef`. The bMech object contains metadata that specifies the following:

- Service Contract: the bMech indicates the PID of the bDef that it is related to. This is like saying that the bMech provides and implementation of the bDef.

- Service binding metadata (i.e., in WSDL) : concrete binding for the **getContent** method that is defined. Specifically, the WSDL indicates that the **getContent** operation binding exists at the base URL of `http://localhost:8080/service/saxon`. Note that this service is hosted at the same host and port as the Fedora repository. As noted earlier, this is a local service that is packaged with Fedora.

- Data input profile that indicates that the bMech service operation **getContent** will take the following inputs at runtime:

  - **"xsl"** with MIME type **text/xml**.

  - **"source"** with MIME type **text/xml**.

7.1.2 Creating a digital object with appropriate datastreams

Now you need to create the new digital object with a disseminator based on this bDef and bMech. To get started follow the same procedure as illustrated in Figure 3, this time entering **demo:300** as the datastream ID and **Example 3** as the Label.

You now need to add the two datastreams, the xml source document and the xsl transform document. Using the same method described in Example 1, select the Datastreams tab and:

- Add a datastream with:
  - **ID** - **Source**
  - **Label** - **Poem XML Source**
  - **Mime type – text/xml**
  - **Control Group** - **Managed Content**
  - **Import location**: `FEDORA_HOME/userdocs/tutorials/2/example3/poem.xml`
- Add a datastream with:
  - **ID** - **Xsl**
  - **Label** - **Poem XSL Transform**
  - **Mime type – text/xml**
  - **Control Group** - **Managed Content**
  - **Import location**: `FEDORA_HOME/userdocs/tutorials/2/example3/poem.xsl`

- Exit the object editor. Next, go to a browser and look at this object via the following URL:

http://localhost:8080/fedora/get/demo:300

7.1.3 Creating the Disseminator

In Fedora Administrator, select **File/Open** and enter **demo:300**. Select the **Disseminators** tab from the digital object window. The resulting dialog will now allow you to enter a new disseminator. Follow these steps:

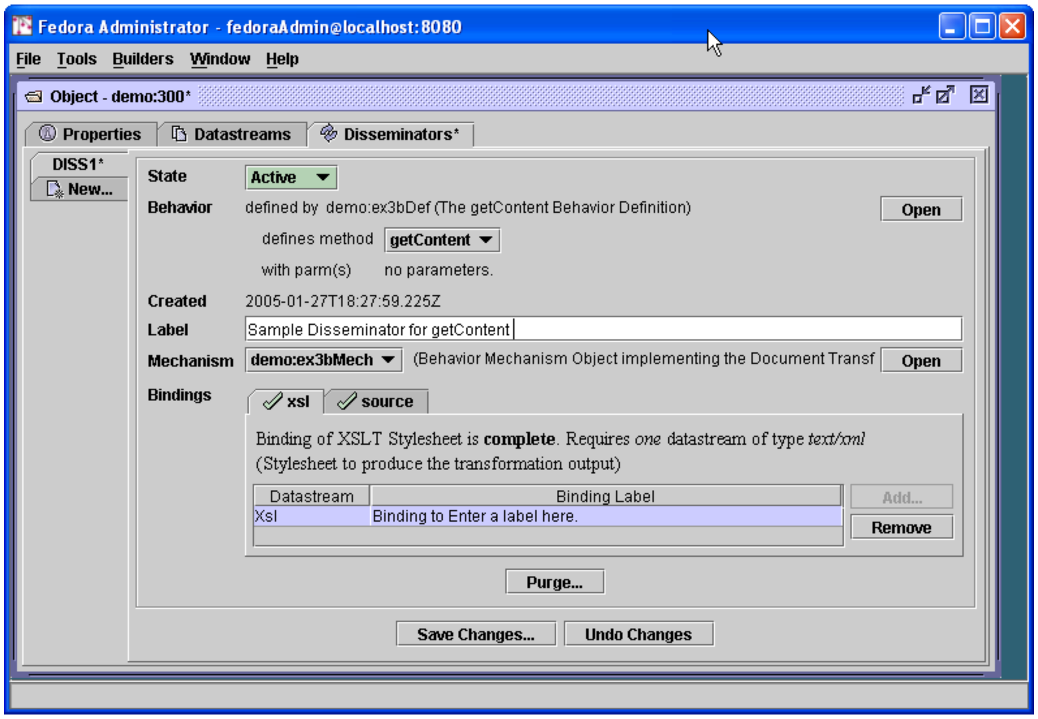
- The **Label** field is just documentation – you can enter **Example 3 Disseminator**.

- Pull down the menu next to **Behavior is defined by...** and select **demo:ex3bDef** . Depending on what you have previously loaded into your repository, you may see other possible selections in this list. By selecting this bDef you are specifying that you would like to use the client-visible methods (in this case only one method `getContent`) defined in **demo:ex3bDef** as the basis for this disseminator. You will note that following this selection, you will see the method list for this bDef, which is the one method **getContent**.

- Pull down the menu next to **Mechanism**. This will show you the set of bMechs that are based on this bDef and which have their datastream requirements (data profile) fulfilled by this digital object. In this case there should be only one: **demo:ex3bMech**.

- Now you need to bind the datastreams in your digital object to the datastreams required for the bMech. At the bottom of the window you should see a set of tabs next to **Bindings**. These tabs show the datastream parameters defined by the bMech, in this case **xsl** and **source**. Select **xsl** and then the **Add...** button. The list will show the set of datastreams in the digital object with MIME type that matches the bMech parameter. You should select the **XSL** datastream. Do the same for the **source** parameter, in this case selecting the **Source** datastream. Note that the similarity of the names of the digital object datastream and the bMech parameter is only for expository purposes in this example – it is not required.

The resulting **Object** window should look like that illustrated in Figure 15. Make sure to select the **Save Disseminator** button to store your work.



State: Active
Behavior: defined by demo:ex3bDef (The getContent Behavior Definition)
defines method: getContent
with parm(s): no parameters.
Created: 2005-01-27T18:27:59.225Z
Label: Sample Disseminator for getContent
Mechanism: demo:ex3bMech (Behavior Mechanism Object implementing the Document Transf
Bindings

Binding of XSLT Stylesheet is **complete**. Requires *one* datastream of type *text/xml*
(Stylesheet to produce the transformation output)

Datastream	Binding Label
Xsl	Binding to Enter a label here.

Figure 15 – Example 3 disseminator creation

You're done! Figure 16 illustrates the role of this digital object and disseminator in response to a client request. You can go to the digital object header page at `http://localhost:8080/fedora/get/demo:300` and select the **View Dissemination Index** link. Your newly added dissemination should now appear, alongside the primitive behaviors for the object.

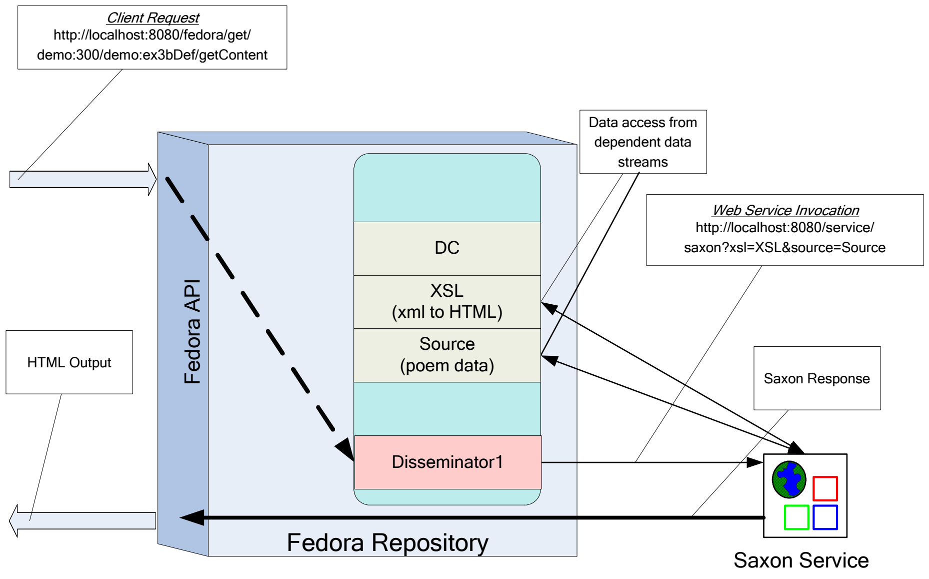

Client Request
http://localhost:8080/fedora/get/
demo:300/demo:ex3bDef/getContent
Data access from dependent data streams
Web Service Invocation
http://localhost:8080/service/
saxon?xsl=XSL&source=Source
Fedora API
DC
XSL
(xml to HTML)
Source
(poem data)
HTML Output
Saxon Response
Disseminator1
Fedora Repository
Saxon Service

Figure 16 - Example 3 Dissemination access

7.2 *Example 4 - Modifying Example 3 using a redirect datastream*

Example 3 packages the XSL transform datastream in the same digital object as the source XML datastream. However, in many cases you will have XSL transform code that you want to share across several XML sources. This section modifies Example 3 to enable this sharing.

This is done by packaging the XSL transform code in a digital object of its own. Then every digital object that needs to make use of the XSL transform code can use the Fedora REST URL to access that datastream. This is done by defining a redirect datastream using the REST URL as the redirect target. Then, the same disseminator design used in Example 3 can be reused. This is known as *dissemination chaining*, whereby the dissemination of one digital object is used by another.

The steps to do this are quite simple and use techniques introduced thus far:

- Create a new digital object (the "XSL" digital object). Assume that the system assigns a PID of **demo:400**. Create one datastream in addition to the DC with ID **XSL**. As before, this datastream should be configured as:
    - **ID - Xsl**
    - **Label - Poem XSL Transform**
    - **Mime type – text/xml**
    - **Control Group - Managed Content**
    - **Import location**: `FEDORA_HOME/userdocs/tutorials/2/example3/poem.xsl`
- Create another digital object (the "disseminator" digital object). Assume assigns the a PID of **demo:500**.
- Create two new datastreams
    - One configured as follows (the same as the **Source** datastream in Example 3):
        - **ID - Source**
        - **Label - Poem XML Source**
        - **MIME type – text/xml**
        - **Control Group - Managed Content**
        - **Import location**: `FEDORA_HOME/userdocs/tutorials/2/example3/poem.xml`

- Now create the datastream that will redirect to the XSL in **demo:400** as follows:
    - **ID - Xsl**
    - **Label - Poem XSL Transform**
    - **Mime Type – text/xml**
    - **Control Group - Redirect**
    - **location**: `http://localhost:8080/fedora/get/demo:400/XSL`

- From the disseminator perspective, this digital object now has the same configuration – two datastreams of MIME type `text/xml`. Thus the same disseminator creation technique that was used in Example 3 can be reused. So, go ahead and follow the Example 3 instructions.

You're done! The **demo:400** digital object should now behave exactly the same as the **demo:300** digital object in Example 3. Figure 17 refines Figure 16 (with some labeling removed for clarity) with the new redirect configuration.

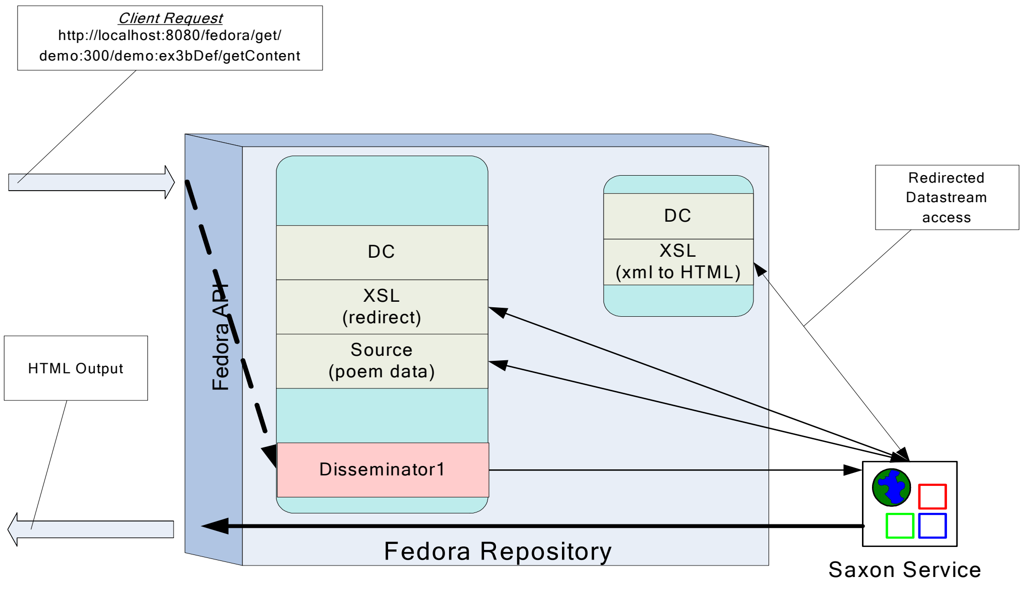


Figure 17 - Dissemintor with redirect datastream

7.3 Example 5 – Writing your own bDefs and bMechs

In order to exploit the full capabilities of Fedora you will need to write your own bDefs and bMechs that can be used as the basis of disseminators. These may exploit existing web services. In addition, you may want to deploy your own web services for use in

Fedora. This section demonstrates how to write a bDef and bMech for an existing web service.

The goal in this example is to construct a digital object containing an image datastream. This digital object will have a disseminator that has two methods: one that converts the image to grayscale and another that resizes the image according to user parameters. The following sections describe the steps to constructing this object. This tutorial will skip over many of the optional documentation-centric aspects of this process, which you will want to use as you develop your repository. You can read about those features in the full Fedora documentation.

7.3.1 The Web Service

This tutorial will not describe how to develop web services. There are many excellent books on that including an on-line tutorial. To use a particular web service in Fedora you only need to know the MIME type of its response and the syntax of the URL that invokes it; i.e., how its arguments are packaged.

A web service invocation URL generally takes the form:

```
http://<base_url>/<operation>?<arg1=val1>&<arg2=val2>...
```

where:

- `<base_url>` includes the host and port of the server that hosts the web service. It also includes a suffix indicating service invocation.
- `<operation>` specifies the particular service operation to be invoked.
- `<arg1=val1&arg2=val2>` is list of argument names and values for the operation.

This example will use two service invocation URLs, with base_url `http://localhost:8080/imagemanip/`, for web services that come packaged with Fedora (note that in the URLs in parentheses denote argument values that must be supplied for invocation of the service):

- `http://localhost:8080/imagemanip/ImageManipulation?op=grayscale&url=(url)` - This operation takes the jpeg image located at the URL `url` and returns a grayscale jpeg image.

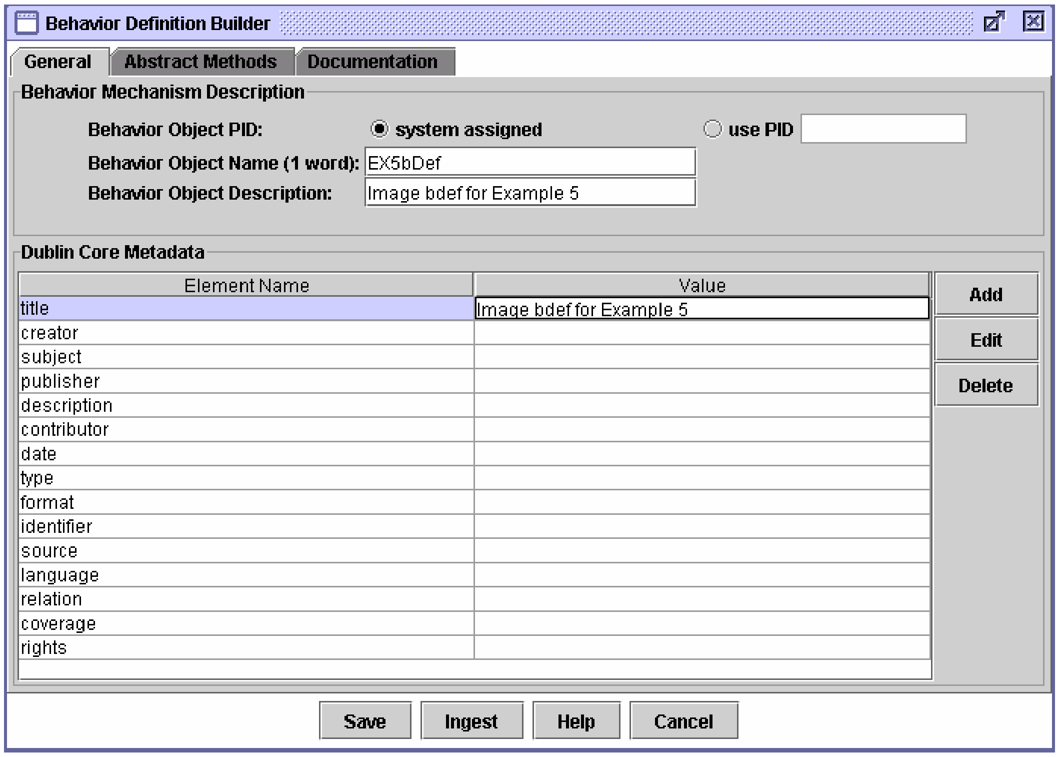
- `http://localhost:8080/imagemanip/ImageManipulation?op=resize&url=(url)&newWidth=(newWidth)` - This operation takes the jpeg image located at the URL `url` and returns a jpeg image resized according to the parameter `newWidth`.

Since your Fedora server is running, you could go to a browser now and invoke these services by directly entering their URLs and appropriate argument values. But you'd have to make sure to encode the `url` argument (which is the URL of the datastream containing the source jpeg image) for inclusion in the full service invocation URL. What a pain! Why not continue with this tutorial section to see how Fedora can do this for you?

7.3.2 Constructing the bDef

As described earlier, a bDef describes client-visible operations and arguments. This specification is independent of the datastream requirements of these methods and their service association. This work is done later in the construction of the bMech.

In the Fedora Administrator select **Builders/Behavior Definition Builder**. Complete the General tab as shown in Figure 18. For the remainder of this description we will assume that the system assigns the PID `demo:600` – your system may assign a different PID to this bDef.



Behavior Object PID: [✓ system assigned] use PID

Behavior Object Name (1 word): EX5bDef

Behavior Object Description: Image bdef for Example 5

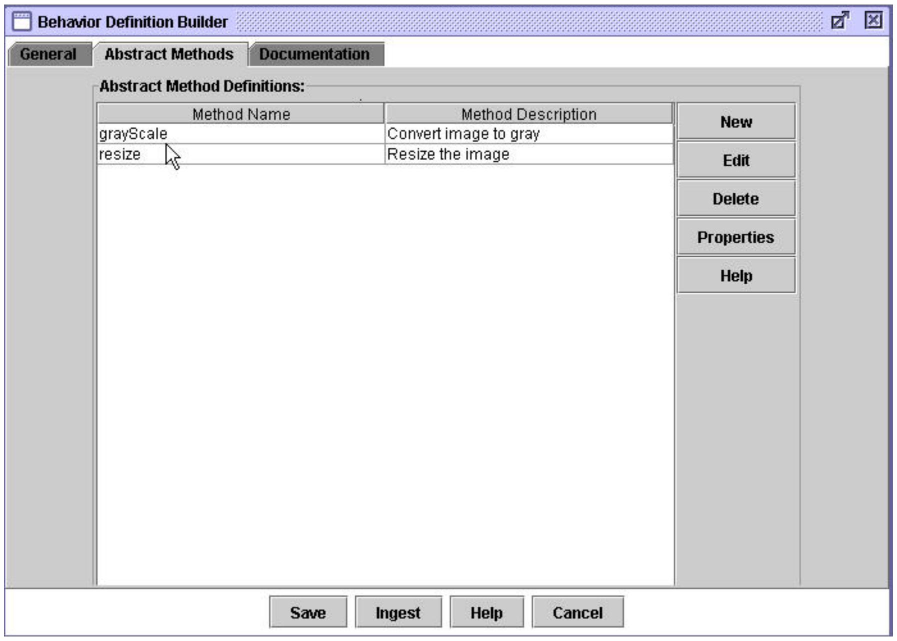
Element Name	Value
title	Image bdef for Example 5
creator	
subject	
publisher	
description	
contributor	
date	
type	
format	
identifier	
source	
language	
relation	
coverage	
rights	

Figure 18 - bDef General Tab

Now select the **Abstract Methods** tab. You will now enter the client visible methods for this bDef. Select the **New** button and, using the **Add Method** window, enter the **Method Name** and **Method Description** for two methods:

- **grayScale** – **Convert image to gray**
- **resize** – **resize the image**

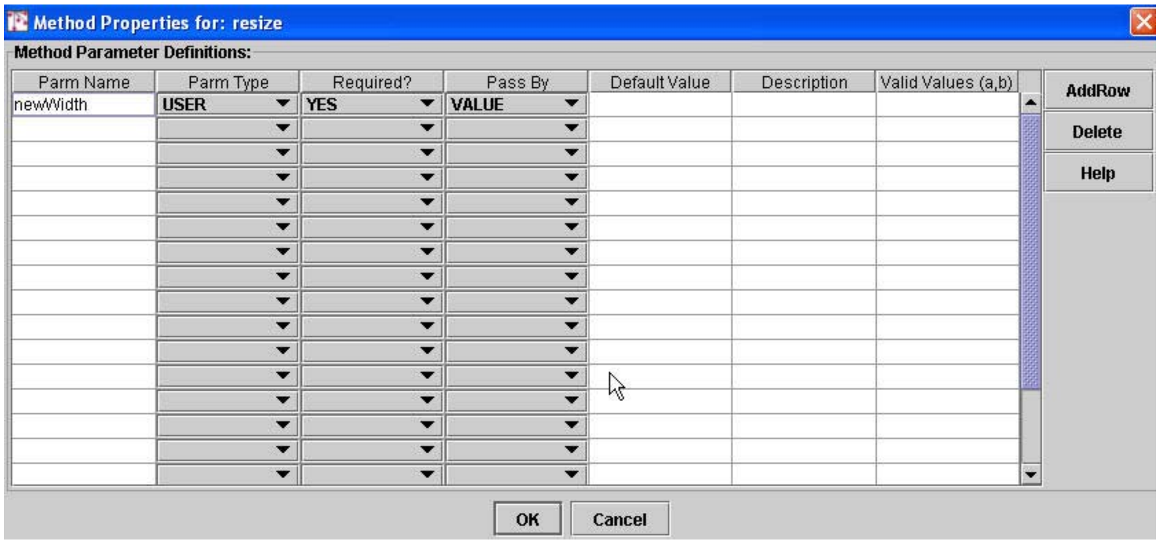
The **Abstract Methods** window should now look like that illustrated in Figure 19.



Method Name	Method Description
grayScale	Convert image to gray
resize	Resize the image

Figure 19 - Abstract Method Definition

The **grayScale** method has no client visible argument. All it does is turn a jpeg to gray. However, the **resize** method requires input from the client specifying the new size. To specify parameters select the **resize** method and then select the **Properties** button. Enter information in this window corresponding to that shown in Figure 20. For the argument **newWidth** you have now specified that the user or client will have to enter the values upon invocation of a disseminator based on this bDef and that the argument value will be passed to the web service.



Parm Name	Parm Type	Required?	Pass By	Default Value	Description	Valid Values (a,b)
newWidth	USER	YES	VALUE			

Figure 20 - Method Parameter Definitions

Select **OK** to return to the main window and select the **Documentation** tag. You can enter some fake values here as:

- **Document Label**: **temp**

- **Document URL**: **http://temp.org**
- **Document MIME Type**: **text/html**

When you develop real applications you will want to fill in real documentation values.

Now select **Ingest** to create your bDef. Your done! You now have a bDef ready to use as the basis of the next step, creating a bMech.

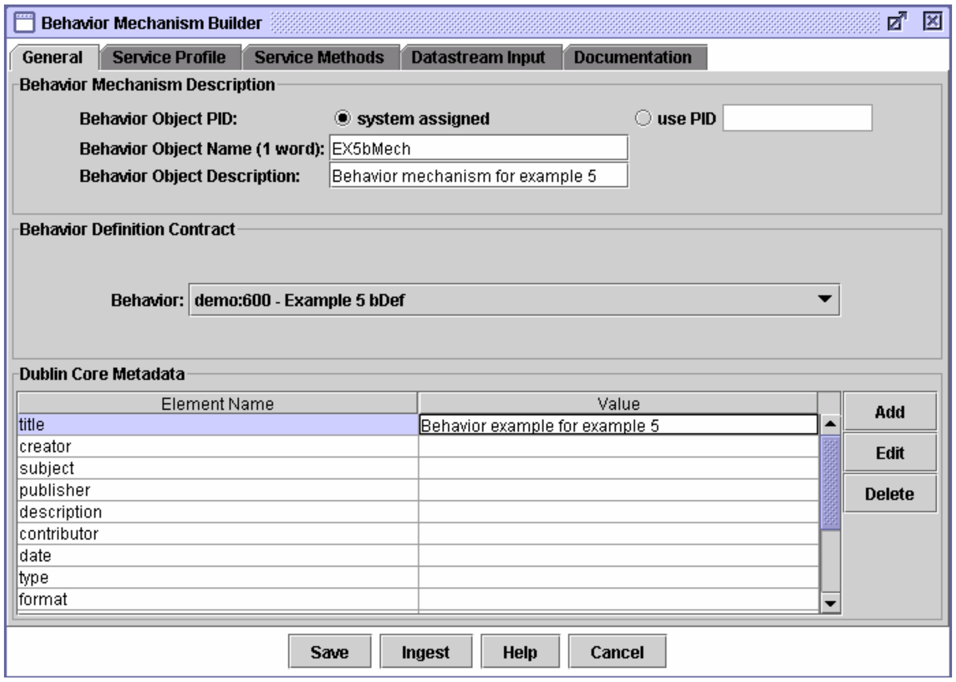
7.3.3 Constructing the bMech

As described earlier a bMech builds on the abstract methods of a bDef by defining:

- Web service call mapping – For each method specified by the bDef, the bMech defines the URL that invokes the web service which is responsible for executing the method.
- Datastream requirements – For each method specified by the bDef, the bMech defines the number and MIME type of datastreams required by the web service to execute that method. As mentioned in Example 3, a bMech may be used as the basis for disseminator for a digital object only if the digital object has the required datastreams. For example if a bMech specifies that it requires one datastream of type `image/jpeg` and one of `text/xml`, it can only be applied to a digital object that as at least one `image/jpeg` datastream and at least one `text/xml` datastream, plus any other types of datastreams.

Note again that there may be many bMechs defined for a bDef, allowing for many digital objects with different configurations that present to the user or client the same interface.

To get started, in the Fedora Administrator select **Builders/Behavior Mechanism Builder**. Complete the General tab as shown in Figure 21. Note the selection for **Behavior Definition Contract**, specifying that this bMech builds on the bDef defined by **demo:600**. This will cause this bMech to inherit the methods defined by **demo:600.** For the remainder of this description we will assume that the system assigns the PID **demo:700** – your system may assign a different PID to this bDef.



Behavior Mechanism Description

Behavior Object PID: [✓ system assigned] use PID

Behavior Object Name (1 word): EX5bMech

Behavior Object Description: Behavior mechanism for example 5

Behavior Definition Contract

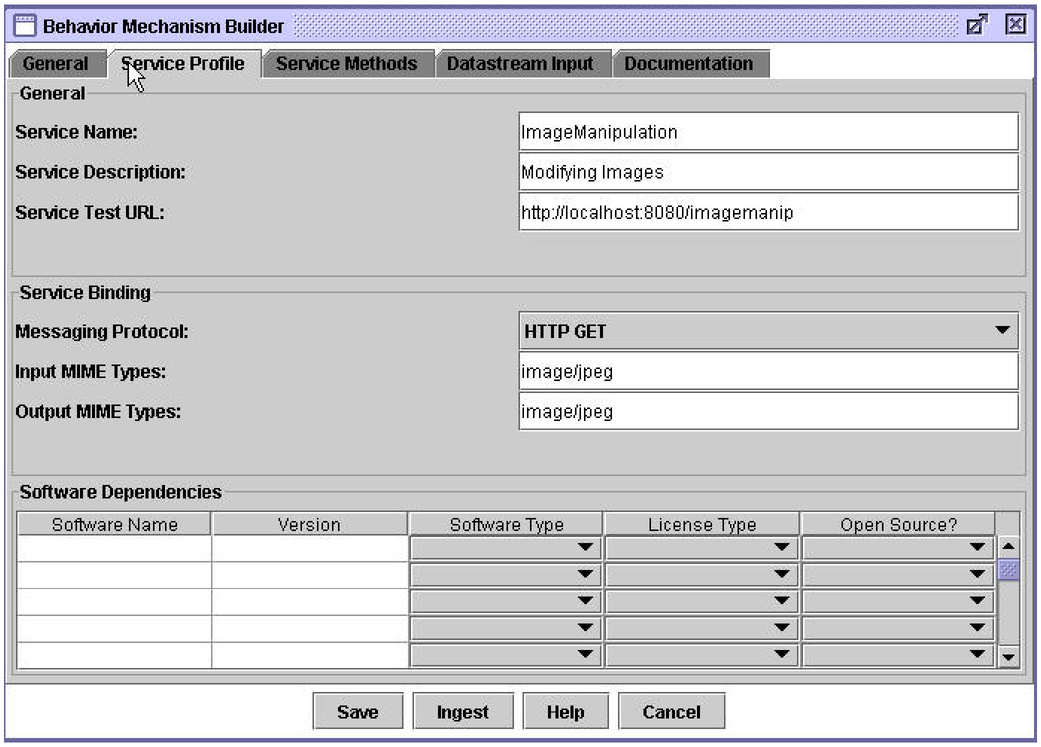
Behavior: demo:600 - Example 5 bDef

Dublin Core Metadata

Element Name	Value
title	Behavior example for example 5
creator	
subject	
publisher	
description	
contributor	
date	
type	
format	

Figure 21 - bMech General Tab

The **Service Profile** tab is used to fill in optional service metadata. For the purpose of this exercise you can fill it in with dummy values as shown in Figure 22. As you develop real applications you will want to complete this with appropriate data.



General

Service Name: ImageManipulation

Service Description: Modifying Images

Service Test URL: http://localhost:8080/imagemanip

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?

Figure 22 - bMech Service Profile

Now select the **Service Methods** tab, where you will complete the bMech specific information for the methods inherited from the bDef. You will notice that these two methods, **grayScale** and **resize**, already appear in this window. Using this window, you should do the following:

- Select the radio button **Base URL:** and fill in the value **http://localhost:8080/imagemanip/.** Refer back to Section 7.3.1 to review the meaning of a base URL, which is the common prefix for the web service calls associated with this bMech.

- Select the first method **grayScale** and then select the **Properties** button. This will show the **Method Properties for: grayScale** window. Follow the following steps (the results of which are shown in Figure 23).

    - Recall from Section 7.3.1 that the web service URL for this operation is `http://localhost:8080/imagemanip/ImageManipulation?op=grayscale&url=(url)`, where the operation name is **ImageManipulation** and there are two parameters, one with name **op** specifying the modification to be performed and the other value that is the URL of the input data. Specify this by filling in the text field next to **HTTP URL (relative):** with **ImageManipulation?op=grayscale&url=(url)** (you will notice that the base URL you entered is already present). The syntax **(url)** indicates that information for this parameter must be entered in the **Method Parameter Definitions** pane (the next step). The other parameter, **op**, does not use this syntax since its value, **grayscale**, is constant.

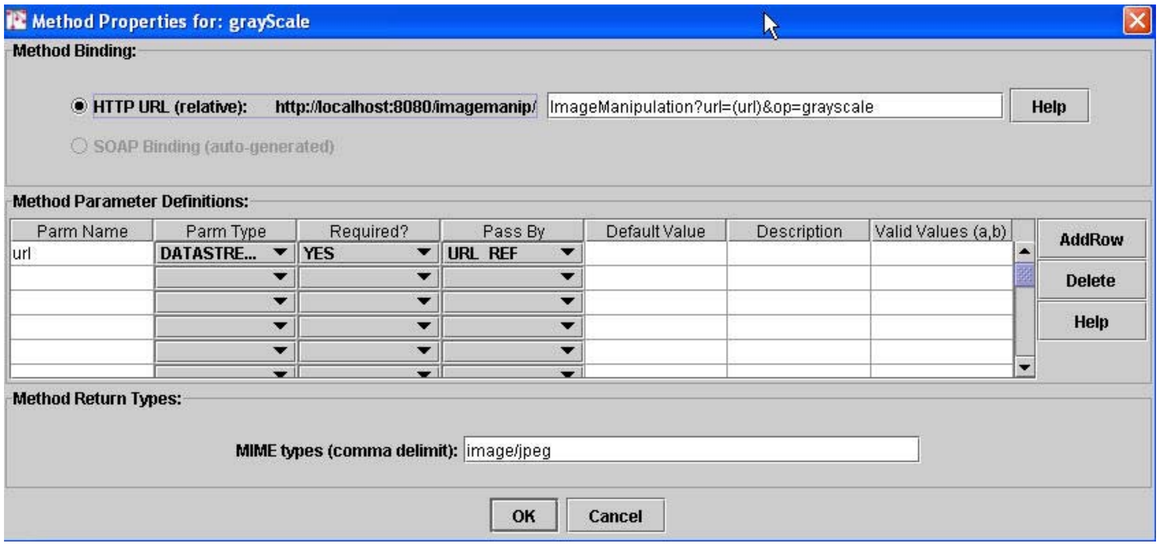
    - In the **Method Parameter Definitions** pane, you need to enter information for each parameter appearing in the web service **HTTP URL** (entered above). In this case, this is only **url** and you should enter the information:

        - **Parm Type** – **DATASTREAM**, indicating that this parameter is a datastream, which will be specified when this bMech is used to establish a disseminator for a digital object.

        - **Required** – **YES**

        - **Pass By** – **URL REF**, indicating that the web service will receive at run-time a URL reference to the respective datastream, which it will access via an HTTP GET request.

    - Lastly, for **MIME types** enter **image/jpeg**, which is the MIME type that the web service will return.


Method Properties for: grayScale
Method Binding:
HTTP URL (relative):
SOAP Binding (auto-generated)
Help
Method Parameter Definitions:
AddRow
Delete
Help
Method Return Types:
OK
Cancel

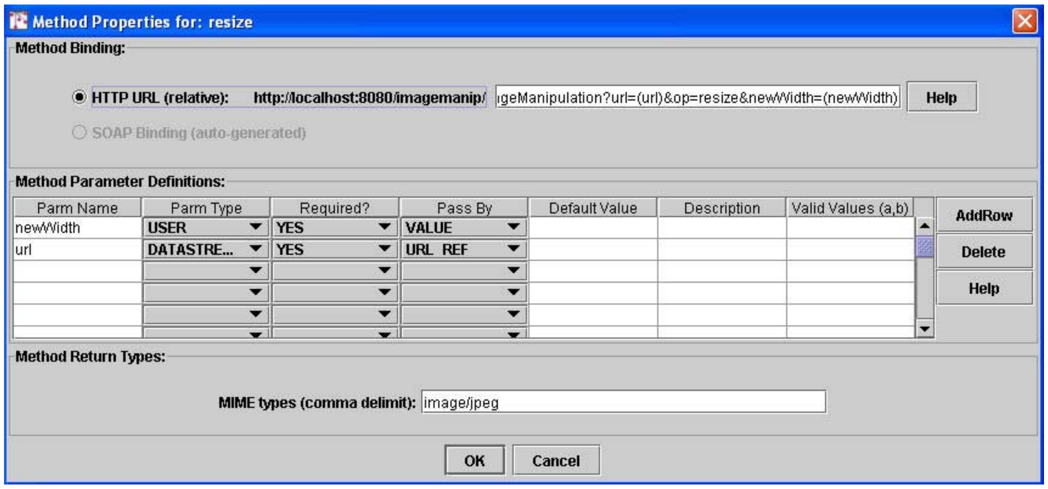
● HTTP URL (relative): http://localhost:8080/imagemanip/ ImageManipulation?url=(url)&op=grayscale

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

MIME types (comma delimit): image/jpeg

Figure 23 - Method Properties window for grayScale

- Return to the **Service Methods** pane by selecting **OK** and select the second method **resize** and then select the **Properties** button. You will now follow the same procedure for defining method parameters, with the following alterations (the results are illustrated in Figure 24):
    - Note that the client parameter defined for the bDef – **newWidth** – is already listed in the parameter list.
    - Complete the **Base URL:** with **ImageManipulation?op=resize&url=(url)&newWidth=(newWidth).** Note now that there are two parameters in the service URL, one of which is particular to this bMech - **(url)** - and one is inherited from the bDef - **(newWidth**). Also note that the correspondence between web service parameter name – e.g., **newWidth** – and the bDef/bMech parameter name **(newWidth)** is not necessary. As before, the syntax **op=resize** indicates a constant argument/value pair that is sent with the service call.
    - The **Method Parameter Definitions** for the bDef defined parameter – **newWidth** – is already completed with information inherited from the bDef. You now must complete the information for **url** in the same manner as you did for the **grayScale** method, since it is used in the same manner.



Method Properties for: resize

Method Binding:

(●) HTTP URL (relative): http://localhost:8080/imagemanip/ ıgeManipulation?url=(url)&op=resize&newWidth=(newWidth) Help

( ) SOAP Binding (auto-generated)

Method Parameter Definitions:

Parm Name	Parm Type	Required?	Pass By	Default Value	Description	Valid Values (a,b)
newWidth	USER	YES	VALUE			
url	DATASTRE...	YES	URL_REF			

Method Return Types:

MIME types (comma delimit): image/jpeg

Figure 24 - Method Properties window for resize

Return to the **Service Methods** pane by selecting **OK.** Select the **Datastream Input** tab. This window shows the parameters with **Parm Type DATASTREAM** that you have entered, in this case **url**. Under **MIMETYPE** you need to complete the MIME type of this datastream – a datastream of this type must appear in a digital object for which this bMech is used to create a disseminator. In this case you should enter **image/jpeg**.

Select the **Documentation** tag. You can enter some fake values here as:

- **Document Label**: **temp**
- **Document URL**: **http://temp.org**
- **Document MIME Type**: **text/html**

When you develop real applications you will want to fill in real documentation values.

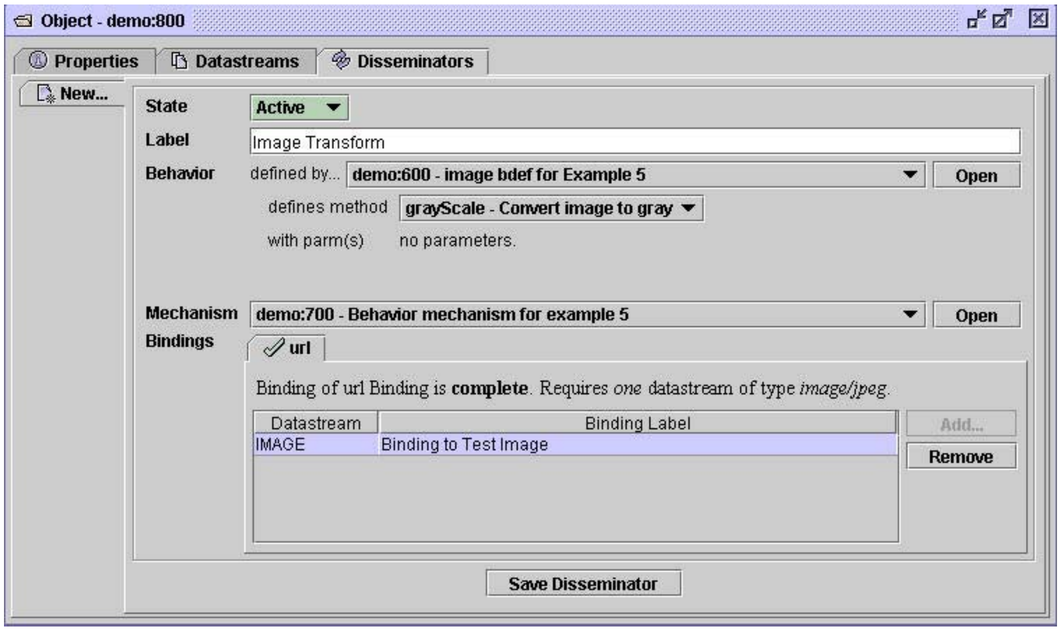
You can now press the **Ingest** button to create this bMech.

You're done! Your newly created bMech is now ready to be used as the basis of a disseminator of a digital object.

7.3.4 Using your bDef and bMech to create a disseminator for an object

bMech creation may have seemed like a lot of work but the result is a reusable object that be the basis of disseminators for many digital objects. Let's use it for one object now. Using the technique you've learned so far, create a new digital object. We'll assume for the remainder of this example that the system assigns the id **demo:800** to it. Add one Managed Content datastream to this digital object with **ID IMAGE** and **MIME type image/jpeg**. Import into this datastream `FEDORA_HOME/server/userdocs/tutorials/2/example4/ex4.jpeg`.

Now add a new disseminator for this digital object. The **Behavior** should be defined by the bDef and the **Mechanism** defined by the bMech created earlier in this section (**demo:600** and **demo:700** in this tutorial, but your PIDs may be different). Recall that in the bMech you defined that a requirement for one datastream parameter: **url** of type **image/jpeg**. You can now establish the **Binding** between that requirement and the **IMAGE** datastream in **demo:800**. The completed disseminator window should like that illustrated in Figure 25.



State: Active
Label: Image Transform
Behavior: defined by... demo:600 - image bdef for Example 5
defines method: grayScale - Convert image to gray
with parm(s): no parameters.
Mechanism: demo:700 - Behavior mechanism for example 5
Bindings

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

Datastream	Binding Label
IMAGE	Binding to Test Image

Figure 25 - Image transformation disseminator

You're done! You can now browse to the header page for this digital object at `http://localhost:8080/fedora/get/demo:800` (the URL may be different depending on your PID) and select the **Dissemination Index** link. You will now find you new disseminator there and can see the effects of the image transformation.

8 What's next?

You should now be ready to create your own bDefs, bMechs, and disseminators of various forms. To explore the other features of Fedora, refer to the full documentation. You can also join the Fedora-users mail list to ask questions and learn from the experience of other Fedora users.