Behavior Mechanisms

A behavior mechanism connects a behavior definition (list of methods) with services that implement the methods.

Writing Behavior Mechanisms

Behavior mechanisms have many restrictions, and it is not always obvious when these restrictions will cause you problems. The best approach is to only use Fedora's internal processing for simple datastream retrievals and single XSL transformations. For anything moderately complex, the mechanism should simply pass the appropriate datastreams to an external service/servlet.

For a mechanism that simply returns a datastream, put the stream name in parentheses (SCREEN), and reference it as a DATASTREAM parameter passed by URL-REF. For a mechanism that performs some operation, enter the URL of the service, adding any data references in parentheses.

When creating a mechanism, you can pass three types of parameters to the target web service:

- DATASTREAM values are refer to datastreams of the object(s) the mechanism will be bound to. They must always be passed by URL-REF, which means you cannot directly access the datastream's value, even if it is just a short string. Contrary to what the Admin tool help system says, you can use DATASTREAM references in a mechanism even though you have defined the mechanism as being "Multi-Server Service".
- DEFAULT values are defined within the mechanism. They are always passed by VALUE, since the data you want entered in the URL is the data you entered. (You could have pasted this information directly into the URL, but that would make it much more difficult to read.) **Note:** Don't declare a default value that points to Fedora on the local server, because this value won't be updated when the object is moved to another repository. Instead, just put the value directly in the service URL.
- USER values must be passed from elsewhere, as part of the URL that calls the dissemination. For an example, see the URL that is created when you use one of the demo image manipulation methods. Note that USER values must be defined in the behavior definition, and have the same name there as they do in the behavior mechanism.

The most frequently used service is Saxon: http://\[hostname\]:\[portNumber\]/saxon/SaxonServlet?source=\[xmlSource\]&style=\[xslStylesheet\]&clear-stylesheet-cache=yes

BMech Tricks and Tips

To pass an object's PID, declare the parameter to be type DEFAULT, but use the default value "$pid" (without quotes). To pass a PID in URI form, use the default value "$objuri".

Although it is possible to pass an object's datastreams to the service the bmech represents, making use of the object's contents within the bmech (before the service is invoked) almost always impossible. You cannot insert the *value* of a datastream directly into the service URL; it can only be passed to the service by reference. You cannot use the value of a dissemination in a mechanism, unless you have created a datastream that redirects to the dissemination. Again, the best you can do is gather up the relevant information and pass it to an external servlet that does the real work.

Disseminators without datastreams: Sometimes, it is useful to write a bmech that doesn't use any datastreams. For example, you may want a utility a mechanism that connects directly to some external site, or one that operates solely on the ResourceIndex. Fedora requires each bmech to reference at least one DATASTREAM parameter, or the Fedora internals will not correctly find/run this behavior. The preferred method of getting around this is to use a datastream called "NULLBIND", which simply maps to the default DC.

Disseminator chaining: You can "chain" disseminators, but it is somewhat clumsy. Object A provides a dissemination. Object B contains a datastream that redirects to A's dissemination. Then B may have a mechanism make use of the datastream, effectively having B's disseminator work off of A's disseminator.

Disseminator redirection: **This trick depends on undocumented Fedora features. It may not work with future releases.** You cannot explicitly mark a disseminator to redirect. By default, disseminators will always pipe their results back through Fedora, so disseminators that result in HTML with relative references will not work correctly. To force a disseminator to redirect, the mechanism should include a dummy datastream that uses the "redirect" content type. This will force the entire disseminator to redirect. (By convention, we use some form of REDIRECT as the name for datastreams of this type.)

Changing behavior mechanisms without changing the API

Fedora doesn't make it easy to modify the definition of a behavior mechanism, but it is possible.

Changing internals of an existing mechanism

Making changes to a mechanism's datastreams can sometimes corrupt the mechanism and/or the database contents, rendering the associated behaviors unusable. Use with care!

In some cases, the internal contents of the mechanism can be changed, but this is difficult because the datastreams generated by the mechanism builder don't have the same structure as the GUI in the mechanism builder. For some reason, most of these datastreams aren't editable in the admin interface, but they can be edited in the raw XML and re-imported. That is, the individual datastreams can be re-imported, but *not* the entire mechanism. A summary of the datastream contents:
DS1 – documentation
WSDL – basic outline of the service **and** the URLs used to fulfill the service.
SERVICE-PROFILE – listing of all input and output formats, not attached to any particular behavior
DC – basic DC data
DSINPUTSPEC – names and MIME types of datastreams this mechanism needs for input
METHODMAP – parameter configuration for all behaviors

Mechanism migration

A relatively safe process for changing mechanisms is to migrate objects between mechanisms. This takes some time, and adds relatively uninformative information to the audit logs, but it's currently the best way to modify mechanisms.

1. Create a new behavior mechansim that does what you want. This is usually accomplished by:
    a. Export a copy of the old mechanism
    b. Find/replace the PID with a new PID, like **bmech:temp**
    c. Make any needed updates
    d. Ingest the new mechanism
2. Identify all objects that use the old mechanism (search for bMech~old_mech_pid)
3. Run a batch-modify script that will replace the old mechanism with the new mechanism. Beware that the disseminator ID may not be consistent across collections, though it should be consistent within a collection.
4. Make sure that the old mechanism is truly gone (search for bMech~old_mech_pid)
5. Purge the old mechanism
6. If you're keeping the new mechanism name, you can stop now (but make sure you update any documentation that refers to the old name). If you want to keep the old mechanism name, start this process over, and change everything to use a *new* new mechanism, which happens to have the same name as the original one.

If the change you're contemplating would impact the behavior definition (API), a similar process should work.