CURL Cookbook - Repository API Examples

This page gives examples of using the curl (documentation here) command on a Unix/Linux or MacOSX system to test the Data Repository. Curl is a very powerful, flexible, command-line HTTP client well-suited to interacting with RESTful Web service APIs like the eagle-i data repository. It has a *lot* of options, so read the doc to understand what they all mean. See the Repository Design Specification and API Manual for the closest thing to an **API reference manual** there is.The commands shown here were tested on Ubuntu Linux 9.10 (Karmic Koala) with the system-provided curl 7.19.5.

Getting Started

In the following examples, the login credentials are given as `"username:password"` – when trying these commands on your own repository, replace those with the real *username* and *password*. These examples assume:

- Your server's URL is **`http://localhost/`**
- Your username is **`username`**
- Your password is **`password`**

..So please adjust as necessary to accomodate your real environment. For example, you may wish to run the curl commands on your MacOS laptop, testing a remote server on http://qa.harvard.eagle-i.net/

1. Test connectivity and show logged-in user (must use GET):

```
curl -G -s -S -u username:password 'http://localhost/repository/whoami?format=text/plain'
```

Creating and Modifying Resource Instances

1. Get a URI for a new instance - note that GET method may not be used. New resource URI is in the output:

```
curl -s -u username:password -d format=text/plain http://localhost/repository/new

(output looks like)
new http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5
```

2. Now create a new instance using that URI, note how it must appear in both the URI arg of the request and as the subject in the inserted statements (which are absolutely bogus and only provided for illustration):

```
curl -s -u username:password -F action=create -F 'insert=@-;type=text/rdf+n3' \
-F 'workspace=http://eagle-i.org/ont/repo/1.0/NG_DefaultWorkspace' \
-F uri=http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5 \
http://localhost/repository/update <<.
<http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5> a <http://eagle-i.org/onto
/#human>;
<http://www.w3.org/2000/01/rdf-schema#label> "Frank N. Furter, PhD";
<http://eagle-i.org/onto/height> "179 cm";
<http://eagle-i.org/onto/mass> "82 kg";
<http://eagle-i.org/onto/hair> "black";
<http://eagle-i.org/onto/eyes> "brown";
<http://eagle-i.org/onto/driversLicense> "MA 12345678".
.
```

3. Display the current state of the instance by resolving the URI in a browser: `http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5`
4. Get a "dissemination" view of the instance, e.g. this request returns N-Triples format:

```
curl -u username:password \
-d format=text/plain http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5
```

5. Make some updates to the instance – *change* the "mass" property to "91 kg", and add a new property "favoriteColor":
   a. First, obtain the edit token:

```
curl -s -u username:password -F action=gettoken -F format=text/plain \
-F uri=http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5 \
http://localhost/repository/update
```

b. Extract the edit token URI from the output (the first element):

```
token   created   creator   new
http://lcs14.intranet.med.harvard.edu/i/00000128-a7b5-5e0d-82b3-624280000000
"2010-05-17T15:20:00.773-04:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
http://serenity/i/0b4565a1-98de-4b3f-9874-7f68e07326b8
"true"^^<http://www.w3.org/2001/XMLSchema#boolean>
```

c. Use that token URI in the request to update the instance. The following command has both a **delete** phase and an **add** phase: First it deletes the old "mass" value (deletes are *always* done first, think about it), and then adds a new "mass" value and a "favoriteColor" value. Note the use of the wildcard URI to delete all values of "mass" without knowing the current objects, `http://eagle-i.org/ont/repo/1.0/MatchAnything`, we can use this because we know (or assume) there is only one value that we want to replace.

```
curl -s -u username:password -F action=update -F 'insert=@-;type=text/rdf+n3' \
--form-string 'token=http://lcs14.intranet.med.harvard.edu/i/00000128-a7b5-5e0d-82b3-624280000000'
\
--form-string 'format=text/rdf+n3' \
--form-string 'delete=<http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-
81931c7151f5> \
<http://eagle-i.org/onto/mass> <http://eagle-i.org/ont/repo/1.0/MatchAnything> .' \
-F uri=http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5 \
http://localhost/repository/update <<.
<http://lcs14.intranet.med.harvard.edu/i/cf8c5f48-cd4d-45be-8245-81931c7151f5> \
<http://eagle-i.org/onto/mass> "91 kg";
<http://eagle-i.org/onto/favoriteColor> "red".
.
```

d. Observe that if you try this update again, or before getting the token, that it is not accepted and the status 409 (Conflict) gets returned. This prevents a "stale" update from undoing the intervening changes.

Managing Named Graphs

The **/repository/graph** service gives you control over the contents of an entire named graph. See the last section of the Admin Manual for an example of how to use this to backup and restore the contents of the entire repository.

1. See the `load-ontology.sh` script in the repository installation kit for an illustration of adding files of serialized RDF to a graph. It has the effect of merging all those files into one graph.
2. Dump out the contents of a graph in, e.g., N3. This gets the NG_Sandbox graph to stdout:

```
curl -X GET -G -s -S -u root:password \
 --data-urlencode 'format=text/rdf+n3' \
 --data-urlencode 'name=http://eagle-i.org/ont/repo/1.0/NG_Sandbox' \
  "http://localhost/repository/graph"
```

3. You can also use **/graph** to do fine-grained editing of the contents of a graph, since it can delete or add a single statement, or a small group, at a time. It is similar to the SPARQL Update protocol in this respect. The following pair of commands *change* the ng-type of the NG_Users named graph from Published to Metadata – this operates on the internals of the repo that you'd normally never see but it is useful to illustrate the point. It uses the N3 serialization format, because it is most convenient for writing your own RDF. Note that unlike the **/update** service, **/graph** cannot delete and insert in one operation.

```
curl -s -S -u root:password  -F 'content=@-;type=text/rdf+n3' \
-F 'name=http://eagle-i.org/ont/repo/1.0/' \
-F 'action=delete' "http://localhost/repository/graph" <<.
<http://eagle-i.org/ont/repo/1.0/NG_Users> <http://eagle-i.org/ont/repo/1.0/ngType>
 <http://eagle-i.org/ont/repo/1.0/NGType_Published> .
.
```

```
curl -s -S -u root:password -F 'content=@-;type=text/rdf+n3' \
-F 'name=http://eagle-i.org/ont/repo/1.0/'  \
-F 'action=add' "http://localhost/repository/graph" <<.
<http://eagle-i.org/ont/repo/1.0/NG_Users>  <http://eagle-i.org/ont/repo/1.0/ngType>
 <http://eagle-i.org/ont/repo/1.0/NGType_Metadata> .
.
```

SPARQL queries

You can script a SPARQL query too, of course. See the API manual for the full range of arguments in the extended SPARQL Protocol the repo supports. Here is an example query that gets a list of all subjects that have a value of the "editor preferred term" predicate, listing each subject once for every predicate so if there are multiple values the subject appears multiple times. This list is used to check for the error of multiple EPTs. Note how curl's form-file-upload mechanism is used to let you enter the lengthy SPARQL query in plain text as an immediate input stream, which is simpler than encoding it as an arg value.

```
curl -s -S -u root:password  -F 'query=@-' -F view=ontology -F format=text/plain \
http://localhost/repository/sparql <<. > terms.txt
SELECT ?subject WHERE {?subject <http://purl.obolibrary.org/obo/IAO_0000111> ?pterm}
order by ?subject
.
```

Maintenance Tasks

Reloading the Repository's Ontology

The repository contains its own internal ontology. Since this is likely to change with each minor release while the repo is under development, *and* the code *depends* on those changes, you may easily find yourself needing to load a new version of the ontology into an existing repo. For example, while upgrading the webapp of a repo while wishing to keep its data intact.The solution is this command that replaces the contents of the named graph containing its ontology. It grovels the ontology right out of the webapp's war file (where it is normally used to bootstrap a new repo). Be sure to replace "**ROOT.war**" with the path to your repo's webapp war file.

```
unzip -p ROOT.war WEB-INF/classes/repository-ont.n3 | curl -s -S -u root:password \
-F 'content=@-;type=text/rdf+n3' \
-F 'name=http://eagle-i.org/ont/repo/1.0/' \
-F 'action=replace' "http://localhost/repository/graph"
```