REST API

- What is DSpace REST API
  - Installing the REST API
    - Disabling SSL
  - REST Endpoints
    - Pagination
    - Index / Authentication
      - Shibboleth Apache configuration for the REST API
    - Communities
    - Collections
    - Items
    - Bitstreams
    - Handle
    - Hierarchy
    - Schema and Metadata Field Registry
    - Report Tools
  - Model - Object data types
    - Community Object
    - Collection Object
    - Item Object
    - Bitstream Object
    - ResourcePolicy Object
    - MetadataEntry Object
    - Schema Object
    - MetadataField Object
    - User Object
    - Status Object
- Introduction to Jersey for developers
- Configuration for DSpace REST
- Recording Proxy Access by Tools
- Additional Information

What is DSpace REST API

The REST API module provides a programmatic interface to DSpace Communities, Collections, Items, and Bitstreams.

DSpace 4 introduced the initial REST API, which did not allow for authentication, and provided only READ-ONLY access to publicly accessible Communities, Collections, Items, and Bitstreams. DSpace 5 builds off of this and allows authentication to access restricted content, as well as allowing Create, Edit and Delete on the DSpace Objects. DSpace 5 REST API also provides improved pagination over resources and searching. There has been a minor drift between the DSpace 4 REST API and the DSpace 5 REST API, so client applications will need to be targeted per version.

Installing the REST API

The REST API deploys as a standard webapp for your servlet container / tomcat. For example, depending on how you deploy webapps, one way would be to alter tomcat-home/conf/server.xml and add:

```
<Context path="/rest"  docBase="/dspace/webapps/rest" />
```

In DSpace 4, the initial/official Jersey-based REST API was added to DSpace. The DSpace 4 REST API provides READ-ONLY access to DSpace Objects.

In DSpace 5, the REST API adds authentication, allows Creation, Update, and Delete to objects, can access restricted materials if authorized, and it requires SSL.

Disabling SSL

For localhost development purposes, SSL can add additional getting-started difficulty, so security can be disabled. To disable DSpace REST's requirement to require security/ssl, alter `[dspace]/webapps/rest/WEB-INF/web.xml` or `[dspace-source]/dspace-rest/src/main/webapp/WEB-INF/web.xml` and comment out the `<security-constraint>` block, and restart your servlet container. Production usages of the REST API should use SSL, as authentication credentials should not go over the internet unencrypted.

REST Endpoints

The REST API is modeled after the DSpace Objects of Communities, Collections, Items, and Bitstreams. The API is not a straight database schema dump of these entities, but provides some wrapping that makes it easy to follow relationships in the API output.

HTTP Header: Accept

Note: You must set your request header's "Accept" property to either JSON (application/json) or XML (application/xml) depending on the format you prefer to work with.

Example usage from command line in XML format with pretty printing:

```
curl -s -H "Accept: application/xml" http://localhost:8080/rest/communities | xmllint --format -
```

Example usage from command line in JSON format with pretty printing:

```
curl -s -H "Accept: application/json" http://localhost:8080/rest/communities | python -m json.tool
```

For this documentation, we will assume that the URL to the "REST" webapp will be http://localhost:8080/rest/ for production systems, this address will be slightly different, such as: https://demo.dspace.org/rest/. The path to an endpoint, will go after the /rest/, such as /rest/communities, all-together this is: http://localhost:8080/rest/communities

Another thing to note is that there are Query Parameters that you can tack on to the end of an endpoint to do extra things. The most commonly used one in this API is "?expand". Instead of every API call defaulting to giving you every possible piece of information about it, it only gives a most commonly used set by default and gives the more "expensive" information when you deliberately request it. Each endpoint will provide a list of available expands in the output, but for getting started, you can start with ?expand=all, to make the endpoint provide all of its information (parent objects, metadata, child objects). You can include multiple expands, such as: ?expand=collections,subCommunities.

Pagination

DSpace 6.x supports pagination by allowing two query parameters: `limit` and `offset`. Note however that the aggregate results number is not supplied (see DS-3887). Endpoints which return arrays of objects, such as /communities, /collections or /items, are "paginated": the full list is broken into "pages" which start at `offset` from the beginning of the list and contain at most `limit` elements. By repeated queries you can retrieve any portion of the array or all of it. The default value of `offset` is 0 and the default value of `limit` is 100. So, as an example, to retrieve the sixth through tenth elements of the full list of Collections, you could do this:

```
curl -s -H "Accept: application/json" http://localhost:8080/rest/collections?offset=5\&limit=5
```

Index / Authentication

REST API Authentication has changed in DSpace 6.x. It now uses a `JSESSIONID` cookie (see below). The previous (5.x) authentication scheme using a `rest-dspace-token` is no longer supported.

Method	Endpoint	Description
GET	/	REST API static documentation page listing available endpoints and their function. *Example*: https://demo.dspace.org/rest

POST	/login	Login to the REST API using a DSpace EPerson (user). It returns a `JSESSIONID` cookie, that can be used for future authenticated requests.
		
		Example Request:
		
		`# Can use either POST or GET (POST recommended). Must pass the parameters "email" and "password".`
		`curl -v -X POST --data "email=admin@dspace.org&password=mypass" https://dspace.myu.edu/rest/login`
		
		Example Response:
		
		`HTTP/1.1 200 OK`
		`Set-Cookie: JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8; Path=/rest/; Secure; HttpOnly`
		
		Example of using JSESSIONID cookie for subsequent (authenticated) requests:
		
		`curl -v --cookie "JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8" https://dspace.myu.edu/rest/status`
		`# This should return <authenticated>true</authenticated>, and information about the authenticated user session`
		
		Invalid email/password combinations will receive an `HTTP 401 Unauthorized` response.
		
		Please note, special characters need to be HTTP URL encoded.
		For example, an email address like `dspacedemo+admin@gmail.com` (notice the + special character) would need to be encoded as `dspacedemo%2Badmin@gmail.com`.

GET	/shibboleth-login	Login to the REST API using Shibboleth authentication. In order to work, this requires additional Apache configuration. To authenticate, execute the following steps:
		
		1. Call the REST Shibboleth login point with a Cookie jar:
		
		`curl -v -L -c cookiejar "https://dspace.myu.edu/rest/shibboleth-login"`
		
		2. This should take you again to the IdP login page. You can submit this form using curl using the same cookie jar. However this is IdP dependant so we cannot provide an example here.
		
		3. Once you submit the form using curl, you should be taken back to the /rest/shibboleth-login URL which will return you the JSESSIONID.
		
		4. Using that JSESSIONID, check if you have authenticated successfully:
		
		`curl -v "http://localhost:8080/dspace-rest/status" --cookie "JSESSIONID=0633C6379266A283E53F65DF8EF61AB9"`

POST	/logout	Logout from the REST API, by providing a `JSESSIONID` cookie. After being posted this cookie will no longer work.
		
		Example Request:
		
		`curl -v -X POST --cookie "JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8" https://dspace.myu.edu/rest/logout`
		
		After posting a logout request, cookie is invalidated and the "/status" path should show you as unauthenticated (even when passing that same cookie). For example:
		
		`curl -v --cookie "JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8" https://dspace.myu.edu/rest/status`
		`# This should show <authenticated>false</authenticated>`
		
		Invalid token will result in HTTP 400 Invalid Request

GET	/test	Returns string "REST api is running", for testing that the API is up.
		
		Example Request:
		
		`curl https://demo.dspace.org/rest/test`
		
		https://demo.dspace.org/rest/test
		
		Example Response:
		
		`REST api is running.`

GET	/status	Receive information about the currently authenticated user token, or the API itself (e.g. version information).
		
		Example Request (XML by default):
		
		`curl -v --cookie "JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8" https://demo.dspace.org/rest/status`
		
		https://demo.dspace.org/rest/status
		
		Example Request (JSON):
		
		`curl -v -H "Accept: application/json" --cookie "JSESSIONID=6B98CF8648BCE57DCD99689FE77CB1B8" https://demo.dspace.org/rest/status`
		
		Example JSON Response:
		
		`{ "okay":true, "authenticated":true, "email":"admin@dspace.org", "fullname":"DSpace Administrator", "sourceVersion":"6.0", "apiVersion":"6" }`

Shibboleth Apache configuration for the REST API

Before Shibboleth authentication for the REST API will work, you need to secure the `/rest/shibboleth-login` endpoint. Add this configuration section to your Apache HTTPD Shibboleth configuration:

```
<Location "/rest/shibboleth-login">
         AuthType shibboleth
         ShibRequireSession On
         # Please note that setting ShibUseHeaders to "On" is a potential security risk.
         # You may wish to set it to "Off". See the mod_shib docs for details about this setting:
         # https://wiki.shibboleth.net/confluence/display/SHIB2/NativeSPApacheConfig#NativeSPApacheConfig-
AuthConfigOptions
         # Here's a good guide to configuring Apache + Tomcat when this setting is "Off":
         # https://www.switch.ch/de/aai/support/serviceproviders/sp-access-rules.html#javaapplications
         ShibUseHeaders On
         require valid-user
</Location>
```

You can test your configuration in 3 different ways:

1. Using a web browser:
    a. Go to `https://dspace.myu.edu/rest/shibboleth-login`, this should redirect you to the login page of your IdP if you don't have a Shibboleth session yet.
    b. Enter your test credentials and this should take you back to the `/rest/shibboleth-login` URL. You should then see a blank page but in the response headers, the JSESSIONID cookie should be present.

c. Then go to `/rest/status` and you should see information on the current authenticated ePerson.

2. Using curl without a Shibboleth Session
   a. Call the REST Shibboleth login point with a Cookie jar:

```
curl -v -L -c cookiejar "https://dspace.myu.edu/rest/shibboleth-login"
```

   b. This should take you again to the IdP login page. You can submit this form using curl using the same cookie jar. However this is IdP dependant so I cannot provide an example here.
   c. Once you submit the form using curl, you should be taken back to the `/rest/shibboleth-login` URL which will return you the JSESSIONID.
   d. Using that JSESSIONID, check if you have authenticated successfully:

```
curl -v "https://dspace.myu.edu/dspace-rest/status" --cookie
"JSESSIONID=0633C6379266A283E53F65DF8EF61AB9"
```

3. Using curl with a Shibboleth Session (cookie)
   a. When you post the Shibboleth login form, the Shibboleth daemon on the **DSpace server** also returns you a Shibboleth Cookie. This cookie looks like `_shibsession_64656661756c74687...` You can also grab this cookie from your browser.
   b. Double check that the cookie you took is valid:

```
curl -v 'https://dspace-url/Shibboleth.sso/Session' -H 'Cookie:
_shibsession_64656661756c7468747470733a2f2f7265706f7369746f72792e636172646966666d65742e61632e756b2f
73686962626f6c657468=_a8d3ad20d8b655250c7357f7ac0e2910;'
```

   c. This should give you information if the Shibboleth session is valid and on the number of attributes.
   d. Use this cookie to obtain a Tomcat JSESSIONID:

```
curl -v 'https://dspace-url/rest/shibboleth-login' -H 'Cookie:
_shibsession_64656661756c7468747470733a2f2f7265706f7369746f72792e636172646966666d65742e61632e756b2f
73686962626f6c657468=_a8d3ad20d8b655250c7357f7ac0e2910;'
```

   e. Use the returned JSESSIONID to check if you have authenticated successfully:

```
curl -v "http://dspace-url/rest/status" --cookie "JSESSIONID=0633C6379266A283E53F65DF8EF61AB9"
```

Communities

Communities in DSpace are used for organization and hierarchy, and are containers that hold sub-Communities and Collections (ex: Department of Engineering).

For an alternative way to create a Community/Collection hierarchy to the REST API take a look at Importing Community and Collection Hierarchy.

HTTP method	REST endpoint	Description
GET	/communities	Return an array of all the communities in the repository. The results are paginated. *Example:* https://demo.dspace.org/rest/communities
GET	/communities/top-communities	Return an array of all top-level communities. The results are paginated. *Example:* *https://demo.dspace.org/rest/communities/top-communities*
GET	/communities/{*community id*}	Return the specified community. *Example:* https://demo.dspace.org/rest/communities/e97b847b-2fd5-4751-8d91-fcf0d8895b81
GET	/communities/{*community id*}/collections	Return an array of collections of the specified community. The results are paginated.

GET	/communities/{*community id*}/communities	Return an array of sub-communities of the specified community. The results are pagi nated.
POST	/communities	Create a new community at top level. You must POST a community object data type.
POST	/communities/{*community id*}/collections	Create a new collections in the specified community. You must POST a collection object data type.
POST	/communities/{*community id*}/communities	Create a new sub-community in the specified community. You must POST a commu nity object data type.
PUT	/communities/{*community id*}	Update the specified community. You must PUT a community object data type.
DELETE	/communities/{*community id*}	Delete the specified community.
DELETE	/communities/{*community id*}/collections/{*collectio n id*}	Delete the specified collection in the specified community.
DELETE	/communities/{*community id*}/communities/{*sub-community id*}	Delete the specified sub-community in the specified community.

Collections

Collections in DSpace are containers of Items. (ex: Engineering Faculty Publications).

For an alternative way to create a Community/Collection hierarchy to the REST API take a look at Importing Community and Collection Hierarchy.

HTTP method	REST endpoint	Description
GET	/collections	Return an array of all the collections in the repository. The results are paginated.
GET	/collections/{*collection id*}	Return the specified collection.
GET	/collections/{*collection id*}/items	Return an array all items of the specified collection. The results are paginated.
POST	/collections/{*collection id*}/items	Create an item in the specified collection. You must POST an item object data type.
POST	/collections/find-collection	Find collection by passed name. Returns the first exact match or nothing. You must POST a single string, not a JSON object.
PUT	/collections/{*collection id*}	Update the specified collection. You must PUT a collection object data type.
DELETE	/collections/{*collection id*}	Delete the specified collection.
DELETE	/collections/{*collection id*}/items/{*item id*}	Delete the specified item in the specified collection.

Items

Items in DSpace represent a "work" and combine metadata and files, known as Bitstreams.

HTTP method	REST endpoint	Description
GET	/items	Return an array of all the items in the repository. The results are paginated. *Example:* https://demo.dspace.org/rest/items
GET	/items/{*item id*}	Return the specified item.
GET	/items/{*item id*}/metadata	Return metadata of the specified item.
GET	/items/{*item id*}/bitstreams	Return an array of all the bitstreams of the specified item. The results are paginated.
POST	/items/find-by-metadata-field	Find items by metadata entry. You must POST a metadataentry object data type.

POST	/items/{*item id*} /metadata	Add metadata to the specified item. You must POST an array of metadataentry object data type.
POST /GET	/items/{*item id*} /bitstreams?name= {file name}	Add bitstream to the specified item. You must POST the file data and include the name parameter with the value as {*file name*} in the URL posted to. *Optional query parameters*: *description*: A description of the bitstream. *groupId*: Id of group to set item resource policy to. *year*: Year to set embargo date to *month*: Month to set embargo date to *day*: Day of month to set embargo date to *Example*: /items/{*item id*}/bitstreams?name=The%20Children%27s%20Crusade%3A%20A%20Duty-Dance%20with%20Death.pdf&description=All%20this%20happened%2C%20more%20or%20less.&groupID=1969&year=2045&month=2&day=13
PUT	/items/{*item id*} /metadata	Update metadata in the specified item. You must PUT a metadataentry object data type. Each metadata entry that will replace all prior matching metadata entries, i.e. if you submit *n* 'dc.subject' entries all pre-existing 'dc.subject' entries in the item will be deleted and replaced with the *n* entries
DELETE	/items/{*item id*}	Delete the specified item.
DELETE	/items/{*item id*} /metadata	Clear the metadata of the specified item.
DELETE	/items/{*item id*} /bitstreams/{*bitstream id*}	Delete the specified bitstream of the specified bitstream.

Bitstreams

Bitstreams are files. They have a filename, size (in bytes), and a file format. Typically in DSpace, the Bitstream will the "full text" article, or some other media. Some files are the actual file that was uploaded (tagged with bundleName:ORIGINAL), others are DSpace-generated files that are derivatives or renditions, such as text-extraction, or thumbnails. You can download files/bitstreams. DSpace doesn't really limit the type of files that it takes in, so this could be PDF, JPG, audio, video, zip, or other. Also, the logo for a Collection or a Community, is also a Bitstream.

HTTP method	REST endpoint	Description
GET	/bitstreams	Return an array of all the bitstreams in the repository. The results are paginated. *Example*: http://demo.dspace.org/rest/bitstreams
GET	/bitstreams/{bitstream id}	Return the specified bitstream.
GET	/bitstreams/{bitstream id}/policy	Return bitstream policies.
GET	/bitstreams/{bitstream id}/retrieve	Return data of bitstream.
POST	/bitstreams/{bitstream id}/policy	Add policy to item. You must POST a resourcepolicy object data type.
PUT	/bitstreams/{bitstream id}/data	Update the data/file of the specified bitstream. You must PUT the data.
PUT	/bitstreams/{bitstream id}	Update metadata of the specified bitstream. You must PUT a bitstream, does not alter the file/data.
DELETE	/bitstreams/{bitstream id}	Delete the specified bitstream.
DELETE	/bitstreams/{bitstream id}/policy/{policy id}	Delete the specified resource policy of the specified bitstream.

You can access the parent object of a Bitstream (normally an Item, but possibly a Collection or Community when it is its logo) through: /bitstreams/:bitstreamID?expand=parent

As the documentation may state "You must post a ResourcePolicy" or some other object type, this means that there is a structure of data types, that your XML or JSON must be of type, when it is posted in the body.

Handle

In DSpace, Communities, Collections, and Items typically get minted a Handle Identifier. You can reference these objects in the REST API by their handle, as opposed to having to use the internal item-ID.

HTTP method	REST endpoint	Description
GET	/handle/{handle prefix}/{handle suffix}	Returns a Community, Collection, or Item object that matches that handle.

Hierarchy

Assembling a full representation of the community and collection hierarchy using the communities and collections endpoints can be inefficient. Retrieve a lightweight representation of the nested community and collection hierarchy. Each node of the hierarchy contains minimal information (id, handle, name).

HTTP method	REST endpoint	Description
GET	/hierarchy	Retrieve a lightweight representation of the nested community and collection hierarchy.

Schema and Metadata Field Registry

HTTP method	REST endpoint	Description
GET	/registries/schema	Return an array of all the schema in the registry
GET	/registries/schema/{schema prefix}	Return the specified schema
GET	/registries/schema/{schema prefix}/metadata-fields/{element}	Return the metadata field within a schema with an unqualified element name
GET	/registries/schema/{schema prefix}/metadata-fields/{element}/{qualifier}	Return the metadata field within a schema with a qualified element name
POST	/registries/schema/	Add a schema to the schema registry. You must POST a schema object data type.
POST	/registries/schema/{schema prefix}/metadata-fields	Add a metadata field to the specified schema. You must POST a metadatafield object data type.
GET	/registries/metadata-fields/{field id}	Return the specified metadata field
PUT	/registries/metadata-fields/{field id}	Update the specified metadata field
DELETE	/registries/metadata-fields/{field id}	Delete the specified metadata field from the metadata field registry
DELETE	/registries/schema/{schema id}	Delete the specified schema from the schema registry

Note: since the schema object contains no data fields, the following method has not been implemented: PUT /registries/schema/{schema id}

Report Tools

Reporting Tools that allow a repository manager to audit a collection for metadata consistency and bitstream consistency. See REST Based Quality Control Reports for more information or test the Collection Report Tool or Metadata Query Tool on demo.dspace.org.

HTTP method	REST endpoint	Description
GET	/reports	Return a list of report tools built on the REST API
GET	/reports/{nickname}	Return a redirect to a specific report
GET	/filters	Return a list of use case filters available for quality control reporting
GET	/filtered-collections	Return collections and item counts based on pre-defined filters
GET	/filtered-collections/{collection id}	Return items and item counts for a collection based on pre-defined filters
GET	/filtered-items	Retrieve a set of items based on a metadata query and a set of filters

Model - Object data types

Here are all of the data types, not all fields are necessary or supported when posting/putting content, but the output contains this information:

Community Object


```
{
  "id":456,
  "name":"Reports Community",
  "handle":"10766/10213",
  "type":"community",
  "link":"/rest/communities/456",
  "expand":["parentCommunity","collections","subCommunities","logo","all"],
  "logo":null,
  "parentCommunity":null,
  "copyrightText":"",
  "introductoryText":"",
  "shortDescription":"Collection contains materials pertaining to the Able Family",
  "sidebarText":"",
  "countItems":3,
  "subcommunities":[],
  "collections":[]
}
```


Collection Object


```
{
  "id":730,
  "name":"Annual Reports Collection",
  "handle":"10766/10214",
  "type":"collection",
  "link":"/rest/collections/730",
  "expand":["parentCommunityList","parentCommunity","items","license","logo","all"],
  "logo":null,
  "parentCommunity":null,
  "parentCommunityList":[],
  "items":[],
  "license":null,
  "copyrightText":"",
  "introductoryText":"",
  "shortDescription":"",
  "sidebarText":"",
  "numberItems":3
}
```


Item Object

```
{
  "id":14301,
  "name":"2015 Annual Report",
  "handle":"123456789/13470",
  "type":"item",
  "link":"/rest/items/14301",
  "expand":["metadata","parentCollection","parentCollectionList","parentCommunityList","bitstreams","all"],
  "lastModified":"2015-01-12 15:44:12.978",
  "parentCollection":null,
  "parentCollectionList":null,
  "parentCommunityList":null,
  "bitstreams":null,
  "archived":"true",
  "withdrawn":"false"
}
```


Bitstream Object


```
{
  "id":47166,
  "name":"appearance and physiology 100 percent copied from wikipedia.pdf",
  "handle":null,
  "type":"bitstream",
  "link":"/rest/bitstreams/47166",
  "expand":["parent","policies","all"],
  "bundleName":"ORIGINAL",
  "description":"",
  "format":"Adobe PDF",
  "mimeType":"application/pdf",
  "sizeBytes":129112,
  "parentObject":null,
  "retrieveLink":"/bitstreams/47166/retrieve",
  "checkSum":{"value":"62778292a3a6dccbe2662a2bfca3b86e","checkSumAlgorithm":"MD5"},
  "sequenceId":1,
  "policies":null
}
```


ResourcePolicy Object


```
{
  "id":317127,
  "action":"READ",
  "epersonId":-1,
  "groupId":0,
  "resourceId":47166,
  "resourceType":"bitstream",
  "rpDescription":null,
  "rpName":null,
  "rpType":"TYPE_INHERITED",
  "startDate":null,
  "endDate":null
}
```


MetadataEntry Object

```
{
  "key":"dc.description.abstract",
  "value":"This is the description abstract",
  "language": null
}
```

Schema Object

```
{
  "namespace": "example.org/myschema",
  "prefix": "myschema"
}
```

MetadataField Object

```
{
  "description" : "myschema test field",
  "element" : "test",
  "name" : "myschema.test",
  "qualifier": null
}
```

User Object

```
{
  "email":"test@dspace.org",
  "password":"pass"
}
```

Status Object

```
{
  "okay":true,
  "authenticated":true,
  "email":"test@dspace.org",
  "fullname":"DSpace Test User",
  "token":"6d45daaa-7b02-4ae7-86de-a960838fae5c"
}
```

Introduction to Jersey for developers

The REST API for DSpace is implemented using Jersey, the reference implementation of the Java standard for building RESTful Web Services (JAX-RS 1). That means this API should be easier to expand and maintain than other API approaches, as this approach has been widely adopted in the industry. If this client documentation does not fully answer about how an endpoint works, it is helpful to look directly at the Java REST API code, to see how it is implemented. The code typically has required parameters, optional parameters, and indicates the type of data that will be responded.

There was no central ProviderRegistry that you have to declare your path. Instead, the code is driven by annotations, here is a list of annotations used in the code for CommunitiesResource.java:

- @Path("/communities"), which then allows it to be routed to http://localhost:8080/communities, this is then the base path for all the requests within this class.
- @GET, which indicates that this method responds to GET http requests
- @POST, which indicates that this method responds to POST http requests
- @PUT, which indicates that this method responds to PUT http requests
- @DELETE, which indicates that this method responds to DELETE http requests

- @Path("/{community_id}"), the path is appended to the class level @Path above, this one uses a variable {community_id}. The total endpoint would be http://localhost:8080/rest/communities/123, where 123 is the ID.
- @Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML }), this indicates that this request expects input of either JSON or XML. Another endpoint accepts HTML input.
- @PathParam("community_id") Integer communityId, this maps the path placeholder variable {community_id} to Java int communityID
- @QueryParam("userIP") String user_ip, this maps a query param like ?userIP=8.8.4.4 to Java String user_id variable, and user_id == "8.8.4.4"

Configuration for DSpace REST

Property	rest.stats
Example Value	true
Informational Note	Boolean value indicates whether statistics should be recorded for access via the REST API; Defaults to 'false'.

Note that if the logging level (in the logging configuration) is set to DEBUG the REST API will output the entire transaction to the logs. In cases where changes are being made to metadata by automatic processes this can result in several gigabyte sized logs depending on the volume.

Recording Proxy Access by Tools

For the purpose of more accurate statistics, a web-based tool may specify who is using it, by adding parameters to the request:

```
http://localhost:8080/rest/items/:ID?userIP=ip&userAgent=userAgent&xforwardedfor=xforwardedfor
```

If no parameters are given, the details of the HTTP request's sender are used in statistics. This enables tools to record the details of their user rather than themselves.

Additional Information

Additional information can be found in the README for dspace-rest, and in the GitHub Pull Request for DSpace REST (Jersey).

Usage examples can be found at: https://github.com/BrunoNZ/dspace-rest-requests

A tutorial on item deposits, together with Postman examples is available at