REST API

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: http://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 .

Index

Method	Endpoint	Description
GET	/	REST API static documentation page
POST	/login	Login to the REST API using a DSpace EPerson (user). It returns a token, that can be used for future authenticated requests (as a value of the rest-dspace-token request header). Example Request: curl -H "Content-Type: application/json" --data '{"email":"admin@dspace.org", "password":"dspace"}' http://localhost:8080/rest/login Example Response: 1febef81-5eb6-4e76-a0ea-a5be245563a5 Invalid email/password combinations will receive an HTTP 403 Forbidden. The extended tokens are generated and stored in memory, not in the database or on disk. There are no timeouts for these tokens. This means that tokens remain valid as long as DSpace is not restarted. A restart of DSpace will invalidate all extended tokens. If applications re-use a token over multiple calls, especially if they are spread over a potentially longer time window, it is highly recommended that the /status endpoint is called to guarantee that a specific token is still valid. Applications that consume the DSpace REST API have no way of telling when DSpace has been restarted. In the DSpace logs, calls with invalid tokens can often look like anonymous requests being made.
POST	/logout	Logout from the REST API, by providing a header rest-dspace-token. After being posted this token will no longer work. Example Request: curl -X POST -H "Content-Type: application/json" -H "rest-dspace-token: 1febef81-5eb6-4e76-a0ea-a5be245563a5" http://localhost:8080/rest/logout 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 http://localhost:8080/rest/test Example Response: REST api is running.
GET	/status	Receive information about the currently authenticated user token. Example Request: curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" -H "rest-dspace-token: f2f478e2-90f2-4e77-a757-4e838ae94154" http://localhost:8080/rest/status Example Response: {"okay":true,"authenticated":true,"email":"admin@dspace.org","fullname":"DSpace Administrator","token":"f2f478e2-90f2-4e77-a757-4e838ae94154"}

Communities

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

- GET /communities - Returns array of all communities in DSpace.
- GET /communities/top-communities - Returns array of all top communities in DSpace.
- GET /communities/{communityId} - Returns community.
- GET /communities/{communityId}/collections - Returns array of collections of community.
- GET /communities/{communityId}/communities - Returns array of subcommunities of community.
- POST /communities - Create new community at top level. You must post community.
- POST /communities/{communityId}/collections - Create new collections in community. You must post Collection.
- POST /communities/{communityId}/communities - Create new subcommunity in community. You must post Community.
- PUT /communities/{communityId} - Update community. You must put Community
- DELETE /communities/{communityId} - Delete community.
- DELETE /communities/{communityId}/collections/{collectionId} - Delete collection in community.
- DELETE /communities/{communityId}/communities/{communityId2} - Delete subcommunity in community.

Collections

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

- GET /collections - Return all collections of DSpace in array. Use the limit parameter to control items per response (default 100) and offset for paging.
- GET /collections/{collectionId} - Return collection with id.
- GET /collections/{collectionId}/items - Return all items of collection. Use the `limit` parameter to control items per response (default 100) and `off set` for paging.
- POST /collections/{collectionId}/items - Create posted item in collection. You must post an Item
- POST /collections/find-collection - Find collection by passed name.
- PUT /collections/{collectionId} - Update collection. You must put Collection.
- DELETE /collections/{collectionId} - Delete collection from DSpace.
- DELETE /collections/{collectionId}/items/{itemId} - Delete item in collection.

Items

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

- GET /items - Return list of items.
- GET /items/{item id} - Return item.
- GET /items/{item id}/metadata - Return item metadata.
- GET /items/{item id}/bitstreams - Return item bitstreams. Use the `limit` parameter to control items per response (default 100) and `offset` for paging.
- _____. You must post a MetadataEntry.

  Unable to locate Jira server

  for this macro. It may be due to Application Link configuration.
- _____ post an array of MetadataEntry
- POST /items/{item id}/bitstreams - Add bitstream to item. You must post a Bitstream
- PUT /items/{item id}/metadata - Update metadata in item. You must put a MetadataEntry
- DELETE /items/{item id} - Delete item.
- DELETE /items/{item id}/metadata - Clear item metadata.
- DELETE /items/{item id}/bitstreams/{bitstream id} - Delete item 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.

- GET /bitstreams - Return all bitstreams in DSpace. Use the `limit` parameter to control items per response (default 100) and `offset` for paging.
- GET /bitstreams/{bitstream id} - Return 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
- PUT /bitstreams/{bitstream id}/data - Update data/file of bitstream. You must put the data
- PUT /bitstreams/{bitstream id} - Update metadata of bitstream. You must put a Bitstream, does not alter the file/data
- DELETE /bitstreams/{bitstream id} - Delete bitstream from DSpace.
- DELETE /bitstreams/{bitstream id}/policy/{policy_id} - Delete bitstream policy.

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.

- GET /handle/{handle-prefix}/{handle-suffix} - Returns a Community, Collection, or Item object that matches that handle.

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}

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

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