B2SHARE REST API Hands-on Exercises

This document contains several exercises that introduce the core functionality of the B2SHARE REST API.
All exercises are to be done using the B2SHARE training instance.

Prerequisites

To make the exercises the following is needed:

- Internet connection and browser (preferably Firefox or Chrome)
- Python 2.7+ installed
- Packages installed:
- requests
- simplejson
- jsonpatch
- copy

All these packages are present on your account on the training VM. If you are using your own laptop environment, please make sure these are installed. Packages can be installed using the PIP tool, see here on how to install this tool. Use the PIP tool as follows:

```
pip install <package name>
```

Requests

Requirements for each request:

- Request URL and HTTP method (e.g. GET, PUT)
- Object identifiers when loading object states (e.g. record, community)

Optional:

- Additional parameters (e.g. your token)
- Data payloads (e.g. files or text)

After each request, check the status of your request by printing the request object or its status code specifically, for example:

```
>>> r = requests.get('https://trng-b2share.eudat.eu/api/communities')
>>> print r
<Response [200]>
>>> print r.status_code
200
```

If the status code is in the range of 300 to 599, the request failed and needs to be investigated:

```
>>> print r
<Response [404]>
>>> print r.reason
NOT FOUND
```

Handling response data

In general, every request made to the B2SHARE service will return a HTTP response code and a text, independent of whether the request was successful or completely understood.

The response text is always in the JSON format and can be interpreted using the `json` package, e.g.:

```
>>> r = requests.get('https://trng-b2share.eudat.eu/api/communities')
>>> res = json.loads(r.text)
```

or

```
>>> res = r.json()
>>> print res['hits']['total']
12
```

Access token loading

For some of the requests, a personal access token is required. This token can be generated on the B2SHARE website by logging in and navigating to your profile page.

Copy the token to the clipboard and store it in a file:

```
echo "your token here" > token.txt
```

Then load it in your Python session:

```
>>> f = open('token.txt', 'r')
>>> token = f.read().strip()
>>> print token
<token>
```

Obtain upload image

For the record creation exercise, make sure you have a file that can be ingested along with the record. Store it in your current working directory, e.g.:

```
wget https://github.com/EUDAT-Training/B2SHARE-Training/raw/master/api/ess2017/img/EUDATSummerSchool.png
```

Remember the file name of this file, since you'll need it in the exercise.

Data retrieval

B2SHARE stores data in separate objects of which each object represents a record, community or other entity. Each object has its own identifier which uniquely identifies the object and makes it possible to refer to it from other objects and using it in URLs. So to retrieve the data of a specific object, this identifier needs to be added to the URL of the request.

Exercises

The exercises in this section can be done using Python, cURL or directly in the browser. Only the creation of new records cannot easily be done in the browser directly through the API, use Python or cURL for this.

Records (15 min)

In this exercise the information and metadata of a single record will be retrieved and processed. In addition, the file contained in the record is downloaded and checked.

Exercise 1a Retrieve single record info

Retrieve the metadata and other information of a single record. You can choose any record on the training website, or use the record ID shown below.

- Endpoint: `/api/records/<RECORD_ID>`
- Method: GET
- Response: 200

Data:

- `RECORD_ID` : 47077e3c4b9f4852a40709e338ad4620

Tasks:

- Create the URL that retrieves the record data and store in the variable `url`

```
>>> url = 'https://trng-b2share.eudat.eu/api/records/47077e3c4b9f4852a40709e338ad4620'
```

- Use the `requests` package to retrieve the object data and optionally the `simplejson` package to format the output of the response text:

```
>>> import requests, simplejson
```

- Execute the request, check whether the request succeeded and store the response text in the variable `res`

```
>>> r = requests.get(url)
>>> print r
<Response [200]>
>>> print r.text
{
...
}
```

Exercise 1b Check metadata and included files

Investigate the metadata and files contained in the record using the results obtained in the previous exercise.

Use the results of the previous exercise.

Tasks:

- Find the ePIC PID of the record by extracting it from the JSON encoded response text:

```
>>> res = r.json()
>>> print res['metadata']['ePIC_PID']
http://hdl.handle.net/11304/619eda56-100f-43f0-9e72-98a22792eb25
```

- Display the value of the title, author and description metadata fields:

```
>>> print res['metadata']['titles']
[{u'title': u'RDA Foundation Governance Document'}]
>>> print res['metadata']['creators']
[{u'creator_name': u'Research Data Alliance Council'}, {u'creator_name': u'RDA2'}]
>>> print res['metadata']['descriptions']
[{u'description': u'A document describing the high-level structures of the Research Data Alliance Foundation. This do
```

- What can you tell about the structure of each metadata field?
- Is the record open access and currently published?

Exercise 1c Download and compare checksum

Download a file from the record's file bucket and compare its checksum.

- Endpoint: `/api/files/<FILE_BUCKET_ID>/<FILE_KEY>`
- Method: GET
- Response: 200

Data:

- `FILE_BUCKET_ID` : 47077e3c4b9f4852a40709e338ad4620

Tasks:

- Determine the number of files stored in the record and store the file metadata in the variable `f`:

```
>>> print len(res['files'])
1
>>> f = res['files'][0]
```

- Locate the file bucket ID for the file in the metadata and store it in the variable `bucket_id`:

```
>>> print f['bucket']
2686d997-87e2-457f-996e-436bb55a84af
```

- Find the checksum of the file:

```
>>> print f['checksum']
md5:c8afdb36c52cf4727836669019e69222
```

- Download the file to the `download` folder by opening a file handle and writing the content of the earlier request:

```
>>> burl = "https://trng-b2share.eudat.eu/api/files/%s/%s" % (f["bucket"], f["key"])
>>> fout = open(f['key'], 'wb')
>>> rf = requests.get()
>>> fout.write(rf.content)
```

- Generate the checksum of the downloaded file (or directly using the request content) using the `md5` package:

```
>>> import md5
>>> md5 = md5.md5(rf.content).hexdigest()
>>> print md5
c8afdb36c52cf4727836669019e69222
```

- Compare to the checksum provided:

```
>>> print md5 == f['checksum'][4:]
True
```

Communities and metadata schemas (15 min)

In this exercise the metadata and metadata schema of a community is retrieved and processed.

Exercise 2a Retrieve existing communities

Retrieve a list of all defined communities.

- Endpoint: `/api/communities`
- Method: GET
- Response: 200

Tasks:

- Retrieve the communities listing:

```
>>> url = 'https://trng-b2share.eudat.eu/api/communities'
>>> r = requests.get(url)
>>> res = r.json()
>>> print res['hits']['hits']
{
...
}
```

- Determine the number of existing communities

```
>>> print res['hits']['total']
12
```

Exercise 2b Retrieve the records of a community

Retrieve a list of all records published under the EUDAT community. This will be done by displaying the records of the repository, but subject to a query parameter containing the community identifier.

- Endpoint: `/api/records?q=community:<COMMUNITY_ID>`
- Method: GET
- Response: 200

Data:

- `COMMUNITY_ID`：`e9b9792e-79fb-4b07-b6b4-b9c2bd06d095`

Tasks:

- Determine the required parameters for the request (`q=community:<COMMUNITY_ID>`)

```
>>> community_id = 'e9b9792e-79fb-4b07-b6b4-b9c2bd06d095'
>>> params = {'q': 'community:' + community_id}
```

- Execute the request with correct URL:

```
>>> url = 'https://trng-b2share.eudat.eu/api/records'
>>> r = requests.get(url, params=params)
>>> print r
<Response [200]>
>>> res = r.json()
```

- Determine the number of records published under this community:

```
>>> print res['hits']['total']
689
```

- Show the metadata of the first published record:

```
>>> print res['hits']['hits'][0]
{
...
}
```

Exercise 2b Retrieve community metadata schema

Retrieve the metadata schema of the EUDAT community.

- Endpoint: `/api/communities/<COMMUNITY_ID>/schemas/last`
- Method: GET
- Response: 200

Tasks:

- Retrieve the community metadata schema:

```
>>> url = 'https://trng-b2share.eudat.eu/api/communities/' + community_id + '/schemas/last'
>>> r = requests.get(url)
>>> print r
<Response [200]>
>>> res = r.json()
```

- Store the schema basic metadata field definitions in the variable `schema`. These fields are stored in the first element of the 'allOf' field of the 'json_schema' dictionary:

```
>>> schema = res['json_schema']['allOf'][0]
>>> print schema
{
...
}
```

- Display the community-specific fields. These fields are stored in the second element of the 'allOf' field of the 'json_schema' dictionary. These fields won't be addressed in the remainder of the exercises, but are shown for clarity:

```
>>> print res['json_schema']['allOf'][1]['properties']
{
...
}
```

- Are there any community-specific fields that need to be filled in for new records?

Exercise 2c Investigate metadata schema structure

Determine the required metadata fields and investigate the required structure of each field.

Use the result of the previous exercise.

Tasks:

- Determine the number of metadata schema fields defined:

```
>>> print len(schema['properties'])
24
```

- Display all metadata fields defined in the schema:

```
>>> print schema['properties'].keys()
[u'embargo_date', u'contributors', u'community', u'titles', u'descriptions', u'keywords', u'$schema', u'open_access',
```

Please note: the `$schema`, `_files`, `_deposit`, `_oai` and `_pid` are not to be filled in.

- Determine the required metadata schema fields:

```
>>> print schema['required']
[u'community', u'titles', u'open_access', u'publication_state', u'community_specific']
```

- Determine the structure of the author, title and description field:

```
>>> print simplejson.dumps(schema['properties']['titles'], indent=4)
{
    "description": "The title(s) of the uploaded resource, or a name by which the resource is known.",
    "title": "Titles",
...
    "type": "array"
}
```

Etc.

Record creation and data upload (30 min)

In this final exercise, a new draft record is created and prepared for final publication. This includes adding initial metadata, updating it through a patch and adding files to publish. Final step is to change its state from draft to published. Once it is published, a record is visible on the B2SHARE website.

Exercise 3a Create a new draft record

Create a new draft record with some metadata values. The record is to be published under the EUDAT community and will be open access. For this a header is prepared that indicates the type of data that's being sent along with the request. Of course, the metadata to be sent is also prepared.

- Endpoint: `/api/records/`
- Method: POST
- Response: 201

Tasks:

- Prepare the header for the request:

```
>>> header = {"Content-Type": "application/json"}
```

- Prepare the payloads, replace the '' with your name so it can be distinguished from other uploads:

```
>>> metadata = {"titles": [{"title": "EUDAT Summer School 2017 upload of <your name>"}],
                "community": "e9b9792e-79fb-4b07-b6b4-b9c2bd06d095",
                "open_access": True}
```

- Load the token to be sent with the request (in case you haven't done so already), since authentication is required:

```
>>> f = open('token.txt', 'r')
>>> token = f.read().strip()
```

- Prepare the parameters dictionary:

```
>>> params = {"access_token": token}
```

- Execute the draft record creation request and check that it worked. Note: add a trailing slash to the URL (as shown here), otherwise the request won't work currently. Again, data is to be sent as a string, so the metadata is converted using the `json` package:

```
>>> r = requests.post('https://trng-b2share.eudat.eu/api/records/',
                      params={'access_token': token}, data=json.dumps(metadata), headers=header)
>>> print r
<Response [201]>
>>> print r.reason
CREATED
```

Note the different response status code for the request.

- Store the draft record ID in the variable `record_id` (which will differ from what you see below, since it is unique):

```
>>> res = r.json()
>>> record_id = res['id']
>>> print record_id
a766efd2e5d543968fff9dd7bf3783c5
```

- Get the `publication_state` metadata value:

```
>>> print res['metadata']['publication_state']
draft
```

Exercise 3b Upload files

Add files to your newly created draft record.

- Endpoint: `/api/files/<FILE_BUCKET_ID>/<FILE_KEY>`
- Method: PUT
- Response: 200

Tasks:

- Files in records are placed in file buckets attached to a record with a specific `FILE_BUCKET_ID`. This identifier can be extracted from the returned information after creating the draft record in the nested property `files` of the property `links`:

```
>>> filebucket_id = res['links']['files'].split('/')[-1]
>>> print filebucket_id
f90aaf16-6bb0-44af-a345-aa492e10ca0e
```

- Set the header for upload. The content type needs to be set as binary stream, while the server must accept JSON information:

```
>>> header = {"Accept": "application/json", "Content-Type": "application/octet-stream"}
```

- Open the file handle of the file to be uploaded (this can be any file you like), e.g. the EUDAT logo:

```
>>> upload_file = {"file": open('EUDAT-logo2011.jpg', 'rb')}
```

- Prepare access token payload to be sent with the request:

```
>>> params = {'access_token': token}
```

- Execute the put request:

```
>>> r = requests.put(res['links']['files'] + '/EUDAT-logo2011.jpg',
                     files=upload_file, params=params, headers=header)
>>> print r
<Response [200]>
```

- Check whether the file was correctly added to the draft record using the data returned in the response text:

```
>>> res = r.json()
>>> print json.dumps(res, indent=4)
{
    "mimetype": "image/jpeg",
    "updated": "2017-07-04T20:39:12.524379+00:00",
    ...
    "key": "EUDAT-logo2011.jpg",
    "size": 34282
}
```

Exercise 3c Update and complete metadata

Create a JSON patch to update the current metadata values of the draft record. Add a description and setting the required field 'community_specific'. To enable easy discoverability in B2FIND later on, a value for the discipline field is added as well.

- Endpoint: `/api/records/<RECORD_ID>/draft`
- Method: PATCH
- Response: 200

Tasks:

- First copy the current metadata to another variable `metadata_new`:

```
>>> import copy
```

```
>>> metadata_new = copy.deepcopy(metadata)
```

- Why is a deep copy used here?
- Why is a simple dictionary copy insufficient?
- Set the new metadata:

```
>>> metadata_new['descriptions'] = [{"description": "Some description", "description_type": "Abstract"}]
>>> metadata_new['community_specific'] = {}
>>> metadata_new['disciplines'] = ["1.4 → Humanities → Arts"]
>>> print json.dumps(metadata_new, indent=4)
...
```

- Create a JSON patch from the old metadata

```
>>> import jsonpatch
>>> patch = jsonpatch.make_patch(metadata, metadata_new)
>>> print patch
[{"path": "/descriptions", "value": [{"description": "Some description", "description_type": "Abstract"}], "op": "ad
```

- Why is the order of variables in the `make_patch` call important?
- Set the header for the patch request:

```
>>> header = {'Content-Type': 'application/json-patch+json'}
```

- Execute the request command with the serialized patch as the data payload:

```
>>> url = 'https://trng-b2share.eudat.eu/api/records/' + record_id + "/draft"
>>> r = requests.patch(url, data=patch.to_string(), params=params, headers=header)
>>> print r
<Response [200]>
```

Exercise 3d Publish record

Create a final JSON patch to publish your draft record and make it publicly available.

- Endpoint: `/api/records/<RECORD_ID>/draft`
- Method: PATCH
- Response: 200

Tasks:

- Create the simple JSON patch (as a string!) to set the `publication_state` to 'submitted':

```
>>> commit = '[{"op": "replace", "path":"/publication_state", "value": "submitted"}]'
```

- Set the header:

```
>>> header = {'Content-Type': 'application/json-patch+json'}
```

- Create the URL do the request:

```
>>> url = 'https://trng-b2share.eudat.eu/api/records/' + record_id + "/draft"
>>> r = requests.patch(url, data=commit, params=params, headers=header)
>>> print r
<Response [200]>
```

- Check the `publication_state` field:

```
>>> res = r.json()
>>> print res['metadata']['publication_state']
published
```

- Why is the value for publication state 'published' instead of 'submitted'?
- Check the DOI and ePIC PID:

```
>>> print res['metadata']['ePIC_PID']
http://hdl.handle.net/0000/a766efd2e5d543968fff9dd7bf3783c5
>>> print res['metadata']['DOI']
http://doi.org/XXXX/b2share.a766efd2e5d543968fff9dd7bf3783c5
```

Unfortunately, these do not work for records generated on the training site of B2SHARE.

- Check your publication on the website using the record ID!