A short introduction to the API that is used by the MDCS

N. Stenberg

1 Introduction

The MDCS works as a storage point for all structured data and meta data that is derrived from Swedish research on materials for Additive manufacturing (AM).

The MDCS has two ways of interaction, the web interface and the Rest API. A Rest API is basically a couple of dedicated web pages where information can be exchanged as opposed to a "full" API where information exchange is separated from the web server.

2 API background

The API documentation is on the web site

https://amdata.proj.kth.se/docs/api

click on **Help** and chose API documentation.

2.1 Benefits with using the API

- Direct access to data
- can use python!
- can use whatever code You like!!
- save all overhead in files
- can write post/get scripts directly into calculation flows

3 API and python3

Focus is on python3. Please do not use python2, it is not sustainable taday.

RestAPIs have a dedicated python module, **requests**. which needs to be installed. Some other data handling modules are needed too.

3.1 Prerequisites

- requests
- numpy
- json
- xmltodict

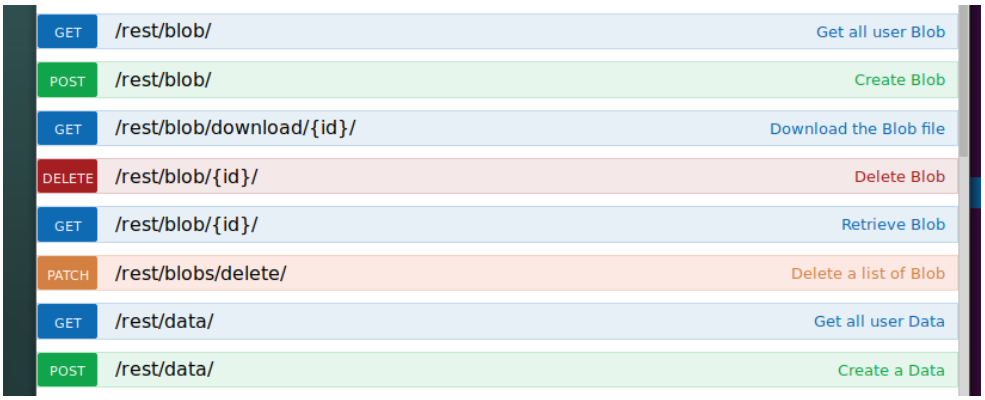

Figure 1: *The documentation of the rest API is a web page where one can try some functions*

- pandas
- io (probably in the default pythin installation)

Use pip or repositories based on preference. Some packages are only available via pip.

If you run windows I think pip is the only way to go.

for pip, based on platform, use either:

```
$ pip install requests numpy json xmltodict pandas
$ pip3 install requests numpy json xmltodict pandas
$ python3 -m pip install requests numpy json xmltodict pandas
```

4 The *requests* function

requests works according to the documentation.

a typical get request looks like:

```
requests.get(URL, data={dict}, auth=(tuple))
```

For *post*, *delete*, *patch* just replace *get*. The output is in some binary json format. If the output is interesting parse the output via the json interpreter:

```
output = json.loads(requests.get('url', data={dict},
         auth=(tuple)).text)
```

4.1 URL

The URL controls what type of request it is. Every specific request all are posted to unique urls. For a full list of what a specific user can do see the API documentation on the MDCS help page.

To post a dataset the URL = '/rest/data'

4.2 data

The data is specified by what type of request it is.

For instance, for a complete list of all the available templates data is not needed and is therefore omitted in the request.

On the other hand, if a template shall be added the *post* command needs to have the template information. That information is included in the data dict.

```
data = {'title': name, 'filename': filename, 'content': tmplStr}
```

4.3 auth

The MDCS is based around users and their accounts. Every post has information of the owner. Therefore every user must be authenticated. When the API is used that needs to be done every time a request is done. auth contain the user name and password as strings in a tuple.

```
auth = ( 'user', 'passwd' )
```

5 The first version of the amdataApi package

A compilation of functions is supplied on github. The function shall be seen as guidance to the API usage. They may be modified according to your own needs. But,

Please! share your improvements! Others might find them useful too.

5.1 How to use

Put the pyApi folder in your PYTHON_PATH or in in your working directory. The ___init___.py file in the folder will make certain that python finds it.

import the functions:

```
from pyApi import amdataApi
```

and you are ready to go.

5.2 The login credentials

First, all functions needed a *authUser* dict thast contain all authentifications including the location of the server.

Each user has its own credentials

```
authUser = { 'url': 'https://amdata.proj.kth.se',
         'user': 'username',
         'passwd': 'password'
       }
```

5.3 The functions in amdataApi

sendBinaryFile

Uploads a file and returns the unique id for that file. Is very useful if the template has a 'blobid' field where it can be referenced.

usage:

```
blobId = amdataApi.sendBinaryFile(filename, authUser)
```

getBinaryFile

Downloads the binary file that is associated with the *dataid* where the field *blobid* exists.

The name of the file that is downloaded will have the name *filename*

usage:

```
amdataApi.getBinaryFile(dataid, filename, authUser)
```

addTemplate

Adds a template to MDCS as global. i.e. available to all. The template must have been correctly formatted prior to sending it the MDCS. See template documentation.

usage:

```
amdataApi.addTemplate('template_filename.xsd', authUser)
```

templateId

Extracts the active *id* for the template with name *tmplName*

If several templates exist with that name, the id for the latest one is returned

usage:

```
tId = amdataApi.templateId(tmplName, authUser)
```

listTemplates(authUser)

Returns two dict with { 'name': 'id' } where

'name' is the given name for that template 'id' is the id-number of the currently used version of the template if an older version is needed use the requests command that is seen here in this function

output:

- the Global templates
- the User's templates

usage:

```
dictG, dictU = amdataApi.listTemplates(authUser)
```

getTemplate(tId, auth):

Returns the content part of the templateId instance as a dict.

The content in that dict needs to be checked because the structure varies dependent on how complex the template is.

usage:

```
tpDict = amdataApi.getTemplate(tId, authUser)
```

Typically the structure is that the posts are listed in either a list of dicts or as just one dict in:

```
tpDict['xsd:schema']['xsd:complexType']
```

from there you are on your own.

listData

Searches for all the values of the desired template posts defined in the list *keys* with an addition of the specific keys: *'id'* and *'title'* which are separated from the template. Returns a list with dicts where the keys values are.

usage:

```
utlist = amdataApi.listData(authUser, keys, templ)
```

getData

Returns the whole data post for one *id* in xml-format

usage:

```
dataXml = amdataApi.getData(dataId, authUser)
```

makeGlobal

Assigns the *dataId* to the Global workspace

usage:

```
amdataApi.makeGlobal(dataId, authUser)
```

sendData

Sends a filled dict(*mdata*) to the MDCS based on the chosen template

required in mdata: mdata['template'] (the given name of the template) bacause the rest of mdata[*] are mapped towards the template posts Returns the response for success check. 200 is OK!, the rest is error codes.

usage:

```
resp = amdataApi.sendData(mdata, authUser)
```

This function is only supplied for syntax checking. How to fill a dataset with data heavily depends on the template. Plaese check the template

saveAndMakeGlobal

Sends the data to the MDCS with the sendData function and makes the data accessible to all.

usage:

```
resp = amdataApi.saveAndMakeGlobal(mdataFull, authUser):
```