Python USI submission REST API Documentation

Release 0.3.1

Paolo Cozzi

Jan 27, 2020

Contents:

CHAPTER 1

Python USI submission REST API

Python USI submission REST API contain all methods to interact with EMBL-EBI Unified Submissions Interface

- Free software: GNU General Public License v3
- Documentation: https://pyusirest.readthedocs.io.

1.1 Features

- Deal with EBI AAP (Authentication, Authorisation and Profile) service, generating tokens and dealing with User and Groups
- Interact with EBI USI (Unified Submission Interface) in order to submit data to biosample as described by this guide. In details:
  - Getting USI API root
  - Selecting a Team
  - Creating a Submission
  - Adding items to Submission
  - Checking Biosample Validation

- Finalising a Submission

1.2 API Endpoints

`pyUSIrest` is written to exploit the BioSamples test environment endpoints. You are incuraged to understand how BioSamples submission works before do a real submission in BioSamples production servers. You can find more information on how to do a real submission in BioSamples production servers in readthedocs documentation: https://pyusirest.readthedocs.io

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Python USI submission REST API, run this command in your terminal:

```
$ pip install pyUSIrest
```

This is the preferred method to install Python USI submission REST API, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Python USI submission REST API can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/cnr-ibba/pyUSIrest
```

Or download the tarball:

```
$ curl -OJL https://github.com/cnr-ibba/pyUSIrest/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Python USI submission REST API in a project, you should import *Root* and *Auth* in order to interact with USI endpoint ad EBI AAP:

```
from pyUSIrest.auth import Auth
from pyUSIrest.usi import Root
```

Warning: Using the BioSamples production endpoints: pyUSIrest is written in order to explot the BioSamples testing environment. You are incuraged to understand the whole process of data submission in the test environment. In order to do a real submission, you have to override the submission endpoints after importing modules before modifying submission objects:

```
pyUSIrest.settings.AUTH_URL = "https://api.aai.ebi.ac.uk"
pyUSIrest.settings.ROOT_URL = "https://submission.ebi.ac.uk"
```

3.1 Creating an Auth object

With an *Auth* object, you're able to generate a AAP token from EBI and use it in browsing USI endpoint. You can instantiate a new *Auth* providing your AAP username and password:

```
auth = Auth(user=<usi_username>, password=<usi_password>)
```

Alternatively you can create an *Auth* object starting from a valid token:

```
auth = Auth(token=<token_string>)
```

3.2 Creating an USI User

In order to create a new USI user, with `pyUSIrest` you can use the method *create_user* of the *User* class:

```
from pyUSIrest.usi import User

user_id = User.create_user(
    user=<new_usi_username>,
    password=<new_password>,
    confirmPwd=<new_password>,
    email=<your_email>,
    full_name=<your full name>,
    organization=<your_organization
)
```

3.3 Creating a Team

To create a team, you will need to create a new *User* from a valid *Auth* object, then you could create a team using the *create_team* method:

```
from pyUSIrest.usi import User
user = User(auth)
team = user.create_team(description="Your description")
```

Warning: remember to ri-generate the token in order to see the new generated team using `pyUSIrest` objects

3.3.1 Add Profile to Domain

Warning: You don't need to do this with a new generated user. You should use this method only if you experience problems when *creating a submission*.

To create a profile for a team:

```
domain = user.get_domain_by_name(<team name>)
domain.create_profile(attributes={"centre name": "My Institution"})
```

For more informations, take a look to creating a domain profile

3.4 Adding User to Team

To add a user to a team, you need to provide a `user_id`, like the one obtained by creating a user, or by calling *get_my_id* from a *User* instance:

```
user = User(auth)
user_id = user.get_my_id()
```

Next, you need to find out the domain reference of a team using a team name and *get_domain_by_name* method:

```
domain = user.get_domain_by_name(team.name)
domain_id = domain.domainReference
```

To add user to a team call *add_user_to_team*:

```
user.add_user_to_team(user_id=user_id, domain_id=domain_id)
```

3.5 Create a Submission

From a valid *Root* object, get the *Team* object providing the `team_name` in which the submission will be created. Then create a new *Submission* using the *create_submission* method:

```
team = root.get_team_by_name(<your team name>)
submission = team.create_submission()
```

If you got a `ConnectionError` exception during last command, you need to add a profile to your domain as described in *add profile to domain*.

3.5.1 Add Samples to a Submission

In order to add sample to a submission, define a `dict` for sample data, then add them using *create_sample*. In the following example, a test animal and a sample from that animal are created:

```
# define data as dictionaries. Ensure that mandatory keys
# are provided or biosample will throw an error
animal_data = {
  'alias': 'animal_1',
  'title': 'A Sample Organism',
  'releaseDate': '2018-06-19',
  'taxonId': 9940,
  'taxon': 'Ovis aries',
  'attributes': {'material': [{'value': 'organism',
    'terms': [{'url': 'http://purl.obolibrary.org/obo/OBI_0100026'}]}],
  'project': [{'value': 'A Sample Project'}]},
  'sampleRelationships': []}

# add this animal to submission
sample = submission.create_sample(animal_data)

# Now generate a sample derived from the previous one.
# This link is provided by sampleRelationships key
sample_data = {'alias': 'sample_1',
  'title': 'A Sample Speciemen',
  'releaseDate': '2018-06-19',
  'taxonId': 9940,
  'taxon': 'Ovis aries',
  'description': 'A Useful Description',
  'attributes': {'material': [{'value': 'specimen from organism',
     'terms': [{'url': 'http://purl.obolibrary.org/obo/OBI_0001479'}]}],
   'project': [{'value': 'A Sample Project'}]},
  'sampleRelationships': [{'alias': 'animal_1',
    'relationshipNature': 'derived from'}]}

# add this sample to the submission
sample = submission.create_sample(sample_data)
```

3.6 Check and finalize a Submission

3.6.1 Querying for biosample validation status

After submitting all data, before finalize a submission, you need to ensure that all the validation steps performed by USI are done with success. You can query status with *get_status*:

```
status = submission.get_status()
print(status)  # Counter({'Complete': 2})
```

status will be a `collections.Counter` object. In order to finalize a submission to biosample, *get_status* need to return only `Complete` as status (not `Pending`), with a number equal to the number of samples attached to submission

3.6.2 Checking errors

Another method to check submission status before finalizing it is to check for errors with *has_errors* method:

```
errors = submission.has_errors()
print(errors)  # Counter({False: 1, True: 1})
```

If there is any `True` in this `collections.Counter` object, submission has errors and can't be finalized. You will need to search for sample with errors in order to remove or update it. Only when this function return `False` with a number equal to the number of attached samples, a submission can be finalized.

3.6.3 Finalize a Submission

After managing sample and validation statuses, if everything is ok you can finalize your submission with *finalize*:

```
submission.finalize()
```

After finalization, you can't add more data to this submission. You may want to reload your data in order to retrieve the *biosample ids*, as described by *get samples from a submission*.

3.7 Fetch a submission by name

In order to get a submission by name, call *get_submission_by_name* from a valid *Root* object:

```
root = Root(auth=auth)
submission = root.get_submission_by_name(
    'c3a7e663-3a37-48d3-a041-8c18088e3185')
```

3.8 Get Sample from a submission

In order to get all samples for a submission, you can call the method *get_samples* on a *Submission* object:

```
samples = submission.get_samples()
```

You can also filter out samples by validationResult or if the have errors or not. For a list of validationResult, check the output of *get_status*:

```
# fetching pending samples
samples_pending = submission.get_samples(validationResult='Pending')

# get samples with errors
samples_error = submission.get_samples(has_errors=True)
```

CHAPTER 4

Advanced Usage

The Python USI submission REST API could be used to manage multiple submissions and samples in the same time. Most of its functions returns iterator objects, in such way some time consiming tasks can be executed lazily and can be filtered and sorted using the appropriate methods. Here there are described some useful tips useful to manage user submission data

4.1 Retrieving Submission Objects

You could retrieve all submission objects from *Root* using *get_user_submissions*: such method returns an iterator object:

```
from pyUSIrest.auth import Auth
from pyUSIrest.usi import Root

auth = Auth(user=<usi_username>, password=<usi_password>)
root = Root(auth)

for submission in root.get_user_submissions():
    print(submission)
```

Submission objects could be also filtered by status or by team name:

```
for submission in root.get_user_submissions(status="Draft", team='subs.test-team-19'):
    print(submission)
```

Submission could be sorted using attributes, as described in Sorting HOW TO:

```
from operator import attrgetter

for submission in sorted(root.get_user_submissions(), key=attrgetter('lastModifiedDate
↪'), reverse=True):
    print(submission)
```

In a similar way, submission could be filtered reling their attributes, for example you can retrieve the recent modified submissions in a similar way:

```
from datetime import datetime, timezone
from dateutil.relativedelta import relativedelta

recent_submission = lambda submission: submission.lastModifiedDate +␣
↪relativedelta(months=+1) > datetime.now(timezone.utc)

for submission in filter(recent_submission, root.get_user_submissions()):
    print(submission)
```

Submission could be derived also from *get_submissions* from a *Team* instance. In this case, the submission will be filtered accordingly to the team, and can be filtered or sorted in the same way as described before:

```
team = root.get_team_by_name('subs.test-team-19')

for submission in team.get_submissions():
    print(submission)
```

4.2 Working with samples

The *get_samples* method from *Submission* returns an iterator of *Sample* instances, and so can be filtered in a similar way as *Submission* instances:

```
submission = root.get_submission_by_name('40549619-7797-4672-b703-93a72c3f984a')

# get all samples in 'Pending' validation status
for sample in submission.get_samples(status="Pending"):
    print(sample)

# get all samples with errors in USI validation
for sample in submission.get_samples(has_errors=True):
    print(sample)

# returning samples with errors in other checks than Ena
for sample in submission.get_samples(has_errors=True, ignorelist=['Ena'])
    print(sample)
```

You can also filter a *Sample* by an attribute, like you can do with *Submission* objects, for example you can retrieve a sample in a submission by title:

```
for sample in filter(lambda sample: sample.title == 'SampleTitle', submission.get_
↪samples()):
    print(sample)
```

CHAPTER 5

pyUSIrest

5.1 pyUSIrest package

5.1.1 Submodules

5.1.2 pyUSIrest.auth module

Created on Thu May 24 15:46:37 2018

@author: Paolo Cozzi <cozzi@ibba.cnr.it>

class pyUSIrest.auth.**Auth**(*user=None*, *password=None*, *token=None*)

Bases: object

Deal with EBI AAP tokens. Starts from a token object or by providing user credentials. It parse token and provide methods like checking expiration times.

auth_url

Default url for EBI AAP.

Type str

expire

when token expires

Type datetime.datetime

issued

when token was requested

Type datetime.datetime

header

token header read by python_jwt.process_jwt

Type dict

claims
token claims read by python_jwt.process_jwt

Type dict

__init__(*user=None*, *password=None*, *token=None*)
Instantiate a new python EBI AAP Object. You can generate a new object providing both user and password, or by passing a valid token string

Parameters

- **user** (*str*) – your aap username
- **password** (*str*) – your password
- **token** (*str*) – a valid EBI AAP jwt token

auth_url = None

get_domains()
Returns a list of domain managed by this object

Returns a list of managed domains

Return type list

get_duration()
Get token remaining time before expiration

Returns remaining time as `timedelta` object

Return type datetime.timedelta

is_expired()
Return True if token is exipired, False otherwise

Returns True if token is exipired

Return type bool

token
Get/Set token as a string

5.1.3 pyUSIrest.client module

Created on Thu Dec 19 16:28:46 2019

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

class pyUSIrest.client.**Client**(*auth*)
Bases: object

A class to deal with EBI submission API. It perform request modelling user token in request headers. You need to call this class after instantiating an *Auth* object:

```
import getpass
from pyUSIrest.auth import Auth
from pyUSIrest.client import Client
auth = Auth(user=<you_aap_user>, password=getpass.getpass())
client = Client(auth)
response = client.get("https://submission-test.ebi.ac.uk/api/")
```

headers
default headers for requests

Type dict

last_response
last response object read by this class

Type requests.Response

last_satus_code
last status code read by this class

Type int

session
a session object

Type request.Session

auth
a pyUSIrest Auth object

Type *Auth*

__init__(*auth*)
Instantiate the class

Parameters **auth** (*Auth*) – a valid *Auth* object

auth
Get/Set *Auth* object

check_headers(*headers=None*)
Checking headers and token

Parameters **headers** (*dict*) – custom header for request

Returns an update headers tocken

Return type headers (dict)

check_status(*response*, *expected_status=200*)
Check response status. See HTTP status codes

Parameters

- **response** (*requests.Reponse*) – the reponse returned by requests
- **method** –

delete(*url*, *headers={}*, *params={}*)
Generic DELETE method

Parameters

- **url** (*str*) – url to request
- **headers** (*dict*) – custom header for request
- **params** (*dict*) – custom params for request

Returns a response object

Return type requests.Response

get(*url*, *headers={}*, *params={}*)
Generic GET method

Parameters

- **url** (*str*) – url to request
- **headers** (*dict*) – custom headers for get request
- **params** (*dict*) – custom params for get request

Returns a response object

Return type requests.Response

headers = {'Accept': 'application/hal+json', 'User-Agent': 'pyUSIrest 0.3.1'}

patch(*url*, *payload={}*, *headers={}*, *params={}*)
Generic PATCH method

Parameters

- **url** (*str*) – url to request
- **payload** (*dict*) – data to send
- **headers** (*dict*) – custom header for request
- **params** (*dict*) – custom params for request

Returns a response object

Return type requests.Response

post(*url*, *payload={}*, *headers={}*, *params={}*)
Generic POST method

Parameters

- **url** (*str*) – url to request
- **payload** (*dict*) – data to send
- **headers** (*dict*) – custom header for request
- **params** (*dict*) – custom params for request

Returns a response object

Return type requests.Response

put(*url*, *payload={}*, *headers={}*, *params={}*)
Generic PUT method

Parameters

- **url** (*str*) – url to request
- **payload** (*dict*) – data to send
- **params** (*dict*) – custom params for request
- **headers** (*dict*) – custom header for request

Returns a response object

Return type requests.Response

class pyUSIrest.client.**Document**(*auth=None*, *data=None*)
Bases: *pyUSIrest.client.Client*

Base class for pyUSIrest classes. It models common methods and attributes by calling *Client* and reading json response from biosample API

_link
`_links` data read from USI response

Type dict

_embeddedd
`_embedded` data read from USI response

Type dict

page
`page` data read from USI response

Type dict

name
name of this object

Type str

data
data from USI read with `response.json()`

Type dict

__init__(*auth=None*, *data=None*)
Instantiate the class

Parameters **auth** (`Auth`) – a valid *`Auth`* object

classmethod clean_url(*url*)
Remove stuff like `{?projection}` from url

Parameters **url** (*`str`*) – a string url

Returns the cleaned url

Return type str

follow_self_url()
Follow *self* url and update class attributes. For instance:

```
document.follow_self_url()
```

will reload document instance by requesting with *`Client.get()`* using `document.data['_links']['self']['href']` as url

follow_tag(*tag*, *force_keys=True*)
Pick a url from data attribute relying on tag, perform a request and returns a document object. For instance:

```
document.follow_tag('userSubmissions')
```

will return a document instance by requesting with *`Client.get()`* using `document._links['userSubmissions']['href']` as url

Parameters

- **tag** (*`str`*) – a key from USI response dictionary
- **force_keys** (*`bool`*) – set a new class attribute if not present

Returns a document object

Return type *Document*

get(*url*, *force_keys=True*)
Override the Client.get method and read data into object:

```
document = Document(auth)
document.get(settings.ROOT_URL + "/api/")
```

Parameters

- **url** (*str*) – url to request
- **force_keys** (*bool*) – If True, define a new class attribute from data keys

Returns a response object

Return type requests.Response

paginate()
Follow all the pages. Return an iterator of document objects

Parameters **response** (*requests.Response*) – a response object

Yields *Document* – a new Document instance

read_data(*data*, *force_keys=False*)
Read data from a dictionary object and set class attributes

Parameters

- **data** (*dict*) – a data dictionary object read with `response.json()`
- **force_keys** (*bool*) – If True, define a new class attribute from data keys

classmethod read_url(*auth*, *url*)
Read a url and returns a *Document* object

Parameters

- **auth** (Auth) – an Auth object to pass to result
- **url** (*str*) – url to request

Returns a document object

Return type *Document*

pyUSIrest.client.**is_date**(*string*, *fuzzy=False*)
Return whether the string can be interpreted as a date.

Parameters

- **string** – str, string to check for date
- **fuzzy** – bool, ignore unknown tokens in string if True

5.1.4 pyUSIrest.exceptions module

Created on Fri Jan 10 16:44:49 2020

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

exception pyUSIrest.exceptions.**NotReadyError**
Bases: RuntimeError

Raised when doing stuff on not ready data (ex finalizing a Submission after validation)

exception pyUSIrest.exceptions.**TokenExpiredError**
Bases: RuntimeError

Raised when token expires while using pyUSIrest

exception pyUSIrest.exceptions.**USIConnectionError**
Bases: ConnectionError

Deal with connection issues with API

exception pyUSIrest.exceptions.**USIDataError**
Bases: Exception

Deal with issues in USI data format

5.1.5 pyUSIrest.settings module

Created on Mon Nov 18 11:47:42 2019

@author: Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

5.1.6 pyUSIrest.usi module

Created on Thu May 24 16:41:31 2018

@author: Paolo Cozzi <cozzi@ibba.cnr.it>

class pyUSIrest.usi.**Domain**(*auth*, *data=None*)
Bases: *pyUSIrest.client.Document*

A class to deal with AAP domain objects

name
domain name

Type str

data
data (dict): data from AAP read with `response.json()`

Type dict

domainName
AAP domainName

Type str

domainDesc
AAP domainDesc

Type str

domainReference
AAP domainReference

Type str

link
`links` data read from AAP response

Type dict

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (Auth) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

create_profile(*attributes={}*)
Create a profile for this domain

Parameters **attributes** (*dict*) – a dictionary of attributes

users
Get users belonging to this domain

class pyUSIrest.usi.**Root**(*auth*)
Bases: *pyUSIrest.client.Document*

Models the USI API Root endpoint

api_root
The base URL for API endpoints

Type str

__init__(*auth*)
Instantiate the class

Parameters **auth** (Auth) – a valid *Auth* object

api_root = None

get_submission_by_name(*submission_name*)
Got a specific submission object by providing its name

Parameters **submission_name** (*str*) – input submission name

Returns The desidered submission as instance

Return type *Submission*

get_team_by_name(*team_name*)
Get a *Team* object by name

Parameters **team_name** (*str*) – the name of the team

Returns a team object

Return type *Team*

get_user_submissions(*status=None*, *team=None*)
Follow the userSubmission url and returns all submission owned by the user

Parameters

- **status** (*str*) – filter user submissions using this status
- **team** (*str*) – filter user submissions belonging to this team

Returns A list of *Submission* objects

Return type list

get_user_teams()
Follow userTeams url and returns all teams belonging to user

Yields *Team* – a team object

class pyUSIrest.usi.**Sample**(*auth*, *data=None*)

Bases: *pyUSIrest.usi.TeamMixin*, *pyUSIrest.client.Document*

A class to deal with USI Samples

alias
The sample alias (used to reference the same object)

Type str

team
team data

Type dict

title
sample title

Type str

description
sample description

Type str

attributes
sample attributes

Type dict

sampleRelationships
relationship between samples

Type list

taxonId
taxon id

Type int

taxon
taxon name

Type str

releaseDate
when this sample will be relased to public

Type str

createdDate
created date

Type str

lastModifiedDate
last modified date

Type str

createdBy
user_id who create this sample

Type str

lastModifiedBy
last user_id who modified this sample

Type str

accession
the biosample_id after submission to USI

Type str

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (*Auth*) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

delete()
Delete this instance from a submission

get_validation_result()
Return validation results for submission

Returns the *ValidationResult* of this sample

Return type *ValidationResult*

has_errors(*ignorelist=[]*)
Return True if validation results throw an error

Parameters **ignorelist** (*list*) – ignore errors in these databanks

Returns True if sample has an errors in one or more databank

Return type bool

patch(*sample_data*)
Update sample by patching data with `Client.patch()`

Parameters **sample_data** (*dict*) – sample data to update

read_data(*data*, *force_keys=False*)
Read data from a dictionary object and set class attributes

Parameters

- **data** (*dict*) – a data dictionary object read with `response.json()`
- **force_keys** (*bool*) – If True, define a new class attribute from data keys

reload()
call `Document.follow_self_url()` and reload class attributes

class pyUSIrest.usi.**Submission**(*auth*, *data=None*)
Bases: *pyUSIrest.usi.TeamMixin*, *pyUSIrest.client.Document*

A class to deal with USI Submissions

id
submission id (*name()* for compatibility)

Type str

createdDate
created date

Type str

lastModifiedDate
last modified date

Type str

lastModifiedBy
last user_id who modified this submission

Type str

submissionStatus
submission status

Type str

submitter
submitter data

Type dict

createdBy
user_id who create this submission

Type str

submissionDate
date when this submission is submitted to biosample

Type str

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (*Auth*) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

check_ready()
Test if a submission can be submitted or not (Must have completed validation processes)

Returns True if ready for submission

Return type bool

create_sample(*sample_data*)
Create a sample from a dictionary

Parameters **sample_data** (*dict*) – a dictionary of data

Returns a *Sample* object

Return type *Sample*

delete()
Delete this submission instance from USI

finalize(*ignorelist=[]*)
Finalize a submission to insert data into biosample

Parameters **ignorelist** (*list*) – ignore samples with errors in these databanks

Returns output of finalize submission as a Document object

Return type *Document*

get_samples(*status=None*, *has_errors=None*, *ignorelist=[]*)
Returning all samples as a list. Can filter by errors and error types:

```
# returning samples with errors in other checks than Ena
submission.get_samples(has_errors=True, ignorelist=['Ena'])

# returning samples which validation is still in progress
submission.get_samples(status='Pending')
```

Get all sample with errors in other fields than *Ena* databank

Parameters

- **status** (*str*) – filter samples by validation status (Pending, Complete)
- **has_errors** (*bool*) – filter samples with errors or none
- **ingnore_list** (*list*) – a list of errors to ignore

Yields *Sample* – a *Sample* object

get_status()
Count validation statues for submission

Returns A counter object for different validation status

Return type collections.Counter

get_validation_results()
Return validation results for submission

Yields *ValidationResult* – a *ValidationResult* object

has_errors(*ignorelist=[]*)
Count sample errors for a submission

Parameters **ignorelist** (*list*) – ignore samples with errors in these databanks

Returns A counter object for samples with errors and with no errors

Return type collections.Counter

name
Get/Set Submission *id*

read_data(*data*, *force_keys=False*)
Read data from a dictionary object and set class attributes

Parameters

- **data** (*dict*) – a data dictionary object read with `response.json()`
- **force_keys** (*bool*) – If True, define a new class attribute from data keys

reload()
call `Document.follow_self_url()` and reload class attributes

status
Return *submissionStatus* attribute. Follow `submissionStatus` link and update attribute is such attribute is None

Returns submission status as a string

Return type str

update_status()
Update *submissionStatus* attribute by following `submissionStatus` link

class pyUSIrest.usi.**Team**(*auth*, *data=None*)
Bases: *pyUSIrest.client.Document*

A class to deal with USI Team objects

name
team name

Type str

data
data (dict): data from USI read with `response.json()`

Type dict

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (`Auth`) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

create_submission()
Create a new submission

Returns the new submission as an instance

Return type *Submission*

get_submissions(*status=None*)
Follows submission url and get submissions from this team

Parameters **status** (*str*) – filter submission using status

Returns A list of *Submission* objects

Return type list

class pyUSIrest.usi.**TeamMixin**
Bases: `object`

__init__()
Instantiate the class

team
Get/Set team name

class pyUSIrest.usi.**User**(*auth*, *data=None*)
Bases: *pyUSIrest.client.Document*

Deal with EBI AAP endpoint to get user information

name
Output of `Auth.claims['nickname']`

Type str

data
data (dict): data from AAP read with `response.json()`

Type dict

userName
AAP username

Type str

email
AAP email

Type str

userReference
AAP userReference

Type str

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (*Auth*) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

add_user_to_team(*user_id*, *domain_id*)
Add a user to a team

Parameters

- **user_id** (*str*) – the required user_id
- **domain_id** (*str*) –

Returns the updated *Domain* object

Return type *Domain*

create_team(*description*, *centreName*)
Create a new team

Parameters

- **description** (*str*) – team description
- **centreName** (*str*) – team center name

Returns the new team as a *Team* instance

Return type *Team*

classmethod create_user(*user*, *password*, *confirmPwd*, *email*, *full_name*, *organisation*)
Create another user into biosample AAP and return its ID

Parameters

- **user** (*str*) – the new username
- **password** (*str*) – the user password
- **confirmPwd** (*str*) – the user confirm password
- **email** (*str*) – the user email
- **full_name** (*str*) – Full name of the user
- **organisation** (*str*) – organisation name

Returns the new user_id as a string

Return type str

get_domain_by_name(*domain_name*)
Get a domain by name

Parameters **domain_name** (*str*) – the required team

Returns the desidered *Domain* instance

Return type *Domain*

get_domains()
Get domains belonging to this instance

Returns a list of *Domain* objects

Return type list

get_my_id()
Get user id using own credentials, and set userReference attribute

Returns the user AAP reference as a string

Return type str

get_team_by_name(*team_name*)
Get a team by name

Parameters **team_name** (*str*) – the required team

Returns the desidered *Team* instance

Return type *Team*

get_teams()
Get teams belonging to this instance

Returns a list of *Team* objects

Return type list

get_user_by_id(*user_id*)
Get a *User* object by user_id

Parameters **user_id** (*str*) – the required user_id

Returns a user object

Return type *User*

user_url = None

class pyUSIrest.usi.**ValidationResult**(*auth*, *data=None*)
Bases: *pyUSIrest.client.Document*

__init__(*auth*, *data=None*)
Instantiate the class

Parameters

- **auth** (Auth) – a valid *Auth* object
- **data** (*dict*) – instantiate the class from a dictionary of user data

has_errors(*ignorelist=[]*)
Return True if validation has errors

Parameters **ignorelist** (*list*) – ignore errors in these databanks

Returns True if sample has errors for at least one databank

Return type bool

pyUSIrest.usi.**check_relationship**(*sample_data*, *team*)
Check relationship and add additional fields if missing

pyUSIrest.usi.**check_releasedate**(*sample_data*)
Add release date to sample data if missing

5.1.7 Module contents

Top-level package for Python EBI submission REST API.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/cnr-ibba/pyUSIrest/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

Python USI submission REST API could always use more documentation, whether as part of the official Python USI submission REST API docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/cnr-ibba/pyUSIrest/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *pyUSIrest* for local development.

1. Fork the *pyUSIrest* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyUSIrest.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyUSIrest
$ cd pyUSIrest/
$ pip install -r requirements_dev.txt
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyUSIrest tests
$ python setup.py test # or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/cnr-ibba/pyUSIrest/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ pytest tests.test_pyUSIrest
```

6.5 Deploying

Current development version is created using:

```
$ bump2version patch # possible: major / minor / patch
```

Other development version (`dev1`, `dev2`) are managed using:

```
$ bump2version build
```

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Paolo Cozzi <paolo.cozzi@ibba.cnr.it>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 TODO

- get a `Team` instance from `Submission` instance
- `Submission.has_errors` make two identical queries, on to determine the status and one to search errors, simplify it by doing only a query
- filtering sample by status or error make a lot of queries. Consider writing coroutines or reading ValidationResult as pages

8.2 0.3.1 (2020-01-27)

- fix a bug when patching a sample: deal with team in relationship
- raise `USIDataError` on `40x` status code
- Change `Auth.__str__()`: now it returns `Token for Foo Bar will expire in HH:MM:SS`
- add `Auth.get_domains` which returns `self.claims['domains']`

8.3 0.3.0 (2020-01-14)

8.3.1 Features

- modelled custom exceptions
- Set a default date if `releaseDate` attribute is missing
- improved documentation by describing how to sort and filter objects
- fix bug when adding samples to a submission retrieved with `team.get_submission()`

- Update documentation. Set `taxon` in sample data (mandatory attribute)
- displaying dates when `print(Submission)` instances
- `Root.get_user_submissions()` and other methods which returned list of objects now return iterator objects
- `str(auth)` will report duration in `hh:mm:ss`
- compiling PDF using PNG images (change badges)
- raise no exceptions where no team is found (using `Root.get_user_teams`)
- Using namespaces to configure API endpoints (`pyUSIrest.settings`)
- move `Root`, `User`, `Domain`, `Team`, `Submission`, `Sample` `ValidationResult` classes inside `pyUSIrest.usi` module

8.4 0.2.2 (2019-03-28)

8.4.1 Features

- Deal with API errors (50x, 40x)

8.5 0.2.1 (2019-01-15)

8.5.1 Features

- test for an empty submission (no samples)
- updated *root.json*, *userSubmission.json* test data
- submissionStatus is no longer an attribute, when feching submission by name is present when getting user submissions
- follow submissionStatus link (if necessary)
- update submission status after create a new submission
- update submission status after `get_submission_by_name`
- update submission status after reload a just finalized submission
- `Domain.users` returns `User` objects in a list
- improved `Submission.get_samples` method

8.6 0.2.0 (2018-10-23)

8.6.1 Features

- Fetch submission by name
- changed library name to `pyUSIrest`
- published to pypi

- Finalize submission with *PUT*

8.7 0.1.0 (2018-10-17)

8.7.1 Features

- submit into biosample with python methods

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

C

D

E

F

G

H

I

L

N

P

R

S

T

U

V