Copernicus Transformation

This document explains how “transform\_copernicus.py” works, the functions it involves and the dependency between different functions.

Inhalt

[Necessary files or folders 2](#_Toc528065876)

[transform\_copernicus.py 2](#_Toc528065877)

[tempjson.json 2](#_Toc528065878)

[pyExcelReader.py 2](#_Toc528065879)

[from\_DOI(f\_doi, doi) 2](#_Toc528065880)

[urlRequest.py 2](#_Toc528065881)

[loginRequest(namePass): 2](#_Toc528065882)

[affRequest(name, ouID\_MPI): 3](#_Toc528065883)

[upfileRequest(Token, filePath, filename): 3](#_Toc528065884)

[itemsRequest(Token, jsonfile): 3](#_Toc528065885)

[itemsSubmit(Token, objId,modiDate): 3](#_Toc528065886)

[subsidiary\_docs (folder that contain mapping files) 3](#_Toc528065887)

[instId\_ctxId.xlsx 3](#_Toc528065888)

[copernicus (folder that contain .xml metadata and .pdf files) 3](#_Toc528065889)

[metadata files: \*.xml 3](#_Toc528065890)

[corresponding attached files: \*.pdf 3](#_Toc528065891)

[copernicus\_DOI\_aff.csv 3](#_Toc528065892)

# Necessary files or folders

## transform\_copernicus.py

This is the main program doing the automatic transformation.

**Transformation idea**:

1. load .xml metadata into a python dict *xmlDict*;
2. load a json metadata template into a python dict *jsondict*;
3. find the values that *jsondict* needed from *xmlDict*.

\*In this transformation program, the matching between keys of xml format and keys of json format should pre-understood.

**Uploading idea**: Via REST API, do login, pdf uploading and item uploading automatically.

The code is elaborately explained in ***transform\_copernicus.py***. Different function blocks are separated by “====” or “----“.

## tempjson.json

This is a template of the json metadata. This file is loaded in ***transform\_copernicus.py*** as a python dict *jsondict.*

## pyExcelReader.py

This file contains two help functions *from\_DOI(f\_doi, doi)* and *pyExlDict(filexl).* (Only *from\_DOI* is used in copernicus’ transformation.)

### from\_DOI(f\_doi, doi)

This function finds ctx\_ID and ou\_ID of the institute to which the publication belongs.

f\_doi is the file that contains mapping from doi to the 3 digit MPI\_ID.

doi is the digital object id of the publication.

A subsidiary file “instID\_ctxID.xlsx” is needed in this function. This file maps MPI\_ID to ctx\_ID and ou\_ID. This file should be located in “subsidiary\_doc” folder. This file is not complete yet (13.09.2018) and should be completed as soon as possible.

## urlRequest.py

This file contains 4 request functions performing automatic interaction with REST API. The functions rely on the python library ‘*requests*’, which is an elegant and simple HTTP library for Python, built for human beings to interact with the webs (for more details please look up to <http://docs.python-requests.org/en/master/> or simply google for it ;).

### loginRequest(namePass):

Login in Pure, get the Token which is necessary for further pdf upload and item upload.

namePass should in the form of “name:password”.

### affRequest(name, ouID\_MPI):

Search for the ou\_ID of the name in Pure via REST API.

Name is a name of an affiliation of an author.

ouID\_MPI is the ou\_ID found by doi mapping.

Elastic search won’t be done for those MPI internal affiliations when the ouID\_MPI is not “xxx”.

The search is done by elastic search. (For more information: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-body.html>, <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html>) Since the elastic search works very “loose” in order to find at least one match for the name, it may cause some mismatch for those names which are not yet in Pure.

### upfileRequest(Token, filePath, filename):

Upload corresponding pdf files of the item via REST API, get the ‘*upfileId*’, which will be the value of the key ‘*content*’ of ‘*files*’ in json format metadata.

In case there is no pdf attached, the function will return “No PDF” for the main program to handle.

### itemsRequest(Token, jsonfile):

Upload the json format metadata *jsonfile* to Pure via REST API.

Call *itemsSubmit()* function to submit the current uploaded item.

### itemsSubmit(Token, objId,modiDate):

Submit the uploaded item to Pure via REST API. The last two parameters comes from the response of item uploading request.

## subsidiary\_docs (folder that contain mapping files)

This folder contains the LUTs (e.g., .xlsx or .csv files) for information mappings.

### instId\_ctxId.xlsx

This file contains the mappings from 3-digit MPI\_ID (column 0) of MPI institutes to their ctx\_ID(column 1, used for item forwarding) and ou\_ID(column 2, used for affiliation links).

This file is relatively consistent and is referred by ***from\_DOI()*** function.

## copernicus (folder that contain .xml metadata and .pdf files)

This folder contains all the .xml metadata to be transformed, the corresponding pdf to be attached and a summary sheet *rsc\_* *copernicus\_DOI\_aff.csv* which is explained below.

### metadata files: \*.xml

### corresponding attached files: \*.pdf

### copernicus\_DOI\_aff.csv

This file contains the mapping from the doi (column 0) of a publication item to the 3-digit MPI\_ID (column 3) of the corresponding MPI institute.

This file is referred by ***transform\_copernicus.py*** as *fileDOIaff* (line 97).

This file should be updated regularly as new .xml metadata files come.