MASI tools for (semi)automated metadata management of historical topographical maps

# Graphical overview



# Workflow

1. Crawl the basic descriptive metadata from the SLUB catalogue with crawlSLUB.py
2. Extract information on georeference from georeferenced map files and query Overpass API for place names inside the maps bounding box with GeoRefInfo.py
3. Join metadata csv tables based on a common key (e.g. filename) using joinTables.py
4. Revision and correction of map specific metadata
5. Create csv file with common metadata shared by all map sheets and general information like contact information etc.
6. Create structured metadata file for every map sheet in yaml format based on a yaml-template using csv2yaml.py
7. Transform the prepared yaml files into ISO standard compliant xml files based on a jinja2-template using yaml2xml.py

# ****crawlSLUB.py****

DESCRIPTION: Tool to collect all information available on the topographic maps from the SLUB web catalogue.

USAGE: Script can be run to save the crawled data as a comma separated table (see below) or it can be imported as a module.

Some manual post-processing/corrections needed in cases where the catalogued records do not follow the common pattern.

RETURNS:

* filename of map
* map series
* sheet number
* sheet name
* information on dates
* scale
* publisher
* date of publication, editing, correction etc.
* online linkage to the map description (landing page)
* online linkage to the map view

INPUT: folder with map files (alternatively list with filenames of maps to crawl)

OUTPUT: metadata\_slub\_maps.csv (metadata table)

log\_SLUB\_unprocessed\_files\_<datestamp>.csv (error log with filenames unable to crawl)

# GeoRefInfo.py

DESCRIPTION: Tool to extract available information on georeference from the (map) files and query the place names for the bounding box area from Overpass API

USAGE: Script can be run to save the data as a comma separated table (see below) or it can be imported as a module to make use of its functions.

RETURNS:

* + filename of map
  + created UUID
  + Reference system authority (e.g. EPSG)
  + Reference system code (e.g. 4314)
  + Bounding Box coordinates (SouthLat, WestLon, NorthLat, EastLon)
  + place names inside the bounding box area of the map

INPUT: folder with **georeferenced** (map) files (geoTiff, geojson, shp)

OUTPUT: metadata\_georef\_maps.csv (metadata table of georeferenced information)

metadata\_places\_maps.csv (metadata table of place name information)

log\_GEOREF\_unprocessed\_files\_<datestamp>.csv (error log with filenames unable to crawl)

log\_PLACE\_unprocessed\_files\_<datestamp>.csv (error log with filenames unable to get Placename for)

# joinTables.py

DESCRIPTION: Tool to join tables based on a common key

USAGE: Script can be run to save the joined table as a comma separated table or it can be imported as a module to make use of its function that returns a joined pandas data frame. Tables are merged to the left (i.e. all values from the first table are preserved – this table is considered to be the complete, where additional values are added to, if present)

RETURNS:

* + pandas data frame with joined columns

INPUT: 2 comma separated tables with a common key (column)

key on the basis of which the join is performed

OUTPUT: comma separated table (;)

and receive place names inside the maps bounding box from OSM Overpass API

# places\_xml.py

= tool to read out place names from already existing metadata xml files (if Overpass request limits are reached or not fast enough)

# main.j2 and contact.j2

= templates of INSPIRE and ISO19139-compliant xml

templating engine used: JINJA2

* a dedicated yaml-file will transform the map specific metadata into the valid xml-schema

# preprocess\_evaluation.py

= tool to preprocess the metadata produced in the segmentation evaluation process (mapping build up area)

Data is saved as a comma separated table, e.g.

* Filename of map
* Shape index
* Area
* Perimeter
* Iteration (of shape analysis procedure)
* Names of training files
* Depth of trees
* Number of test nodes
* Number of trees in the forest
* Width of window
* Number of threads (at the training step)
* Pair parameter for the CRF

IN: Folder with eval<nr> files (files with evaluation metadata for every map)

metadata.legend (Legend/Description of values captured in the evaluation files)

name.list (assignment of map file names to <nr> in evaluation file name)

OUT: parameter\_segm\_evaluation.csv (metadata table of evaluation)

log\_unevaluated\_files.csv (log of not evaluated files)

Mode “Generating”

this mode is used to transform and map simple metadata into standard conform metadata

**Metadata input file (csv):**

**provide the metadata for a number of datasets in one file row by row, with the first row defining field names, fields shall by separated by comma or tab**

**Mapfile:**

1. provide a mapfile to map the field names of your simple metadata .csv-file to element names of your chosen standard

* naming convention: [project name]\_[standard name].csv, e.g*. MASi\_iso19115.csv*
* give the field names of your simple metadata .csv-file in the first column and the standard element names in the second column delimited by “>”, e.g. *id>gmd:identifier*

1. if no mapfile is provided, the field names of the metadata are manually mapped by command line input and saved as a new mapfile following the naming convention (i.e. project name and standard name have to be given as command line arguments)

**Metadata output file (xml):**

**one standard conform xml file is created for every dataset (row) specified in the metadata input file (csv)**

**the xml is named after a given identifier**