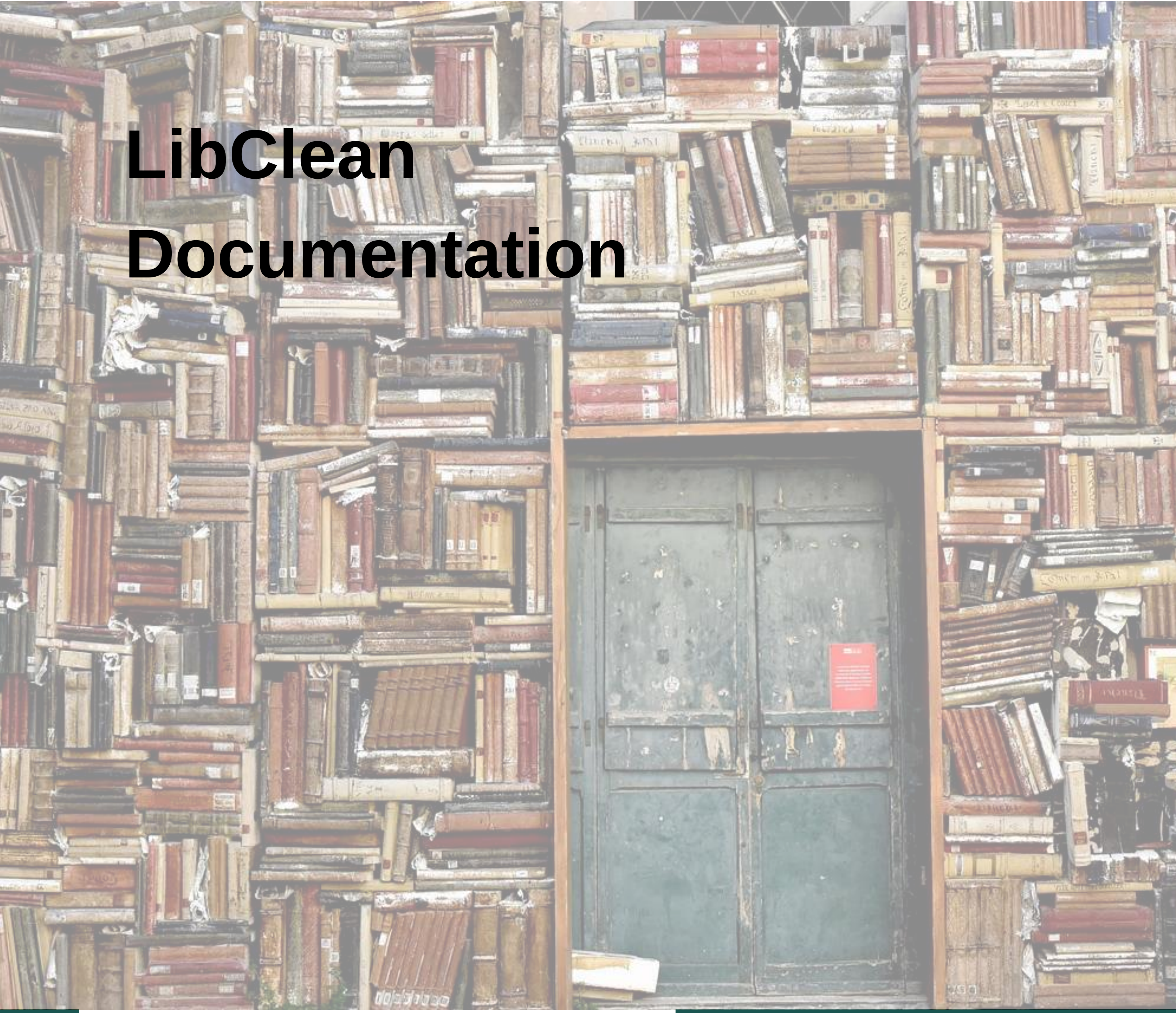
# LibClean Documentation



**Author: Max Collins**

**GitHub: https://github.com/maxcollins1999**

# Contents

# 1 Introduction to LibClean

LibClean is a python [Python Core Team, 2020] package designed to simplify the process of geolocating images based on their metadata. LibClean was intended to meet the requirements of the OldPerth website, which displays historic images from around Perth on an interactive map. The OldPerth website required image geolocation, a persistent image catalogue and client-side website update capabilities.

Data must be provided to LibClean in the form of a MARC 21 formatted XML file, which is then read, and the relevant metadata fields parsed. To do this LibClean utilities pymarc [Summers, 2019], a python package capable of reading MARC 21 formatted data. Once the metadata has been parsed, LibClean uses a combination of pattern matching and list searching to identify and extract possible street addresses.

Image geolocation is accomplished through the use of the Mappify API. An API query is made for each image with an extracted street address. If the street address is valid, Mappify will return the latitude, longitude coordinates, which are then parsed. If the street address is invalid Mappify will return null values and LibClean will denote the corresponding image as having no valid address.

LibClean can save the currently loaded image catalogue as a JSON file. The saved JSON files can then be loaded and merged with new catalogues, with LibClean automatically removing duplicated entries. LibClean is able to push/pull the JSON files to/from a google drive. By saving state to a google drive LibClean creates a persistent catalogue of images, remotely accessible by anyone with the credentials to the host google drive.

Updates to the OldPerth website are accomplished via FTP with SSL explicit encryption. LibClean generates the files required to update the site's image catalogue, and given the correct server name and password, pushes the update.

# 2 Getting Started

## 2.1 Dependencies

LibClean was built in the Anaconda distribution of Python 3.7.4 [Anaconda, 2020].

**Requires:**

- Python ≥ 3.6
- tqdm ≥ 4.36.1
- requests ≥ 2.22.0
- pydrive ≥ 1.3.1

## 2.2 Installation

To install LibClean, fork the repository https://github.com/maxcollins1999/LibClean. Then from the "LibClean - Release" directory, run the following in the console:

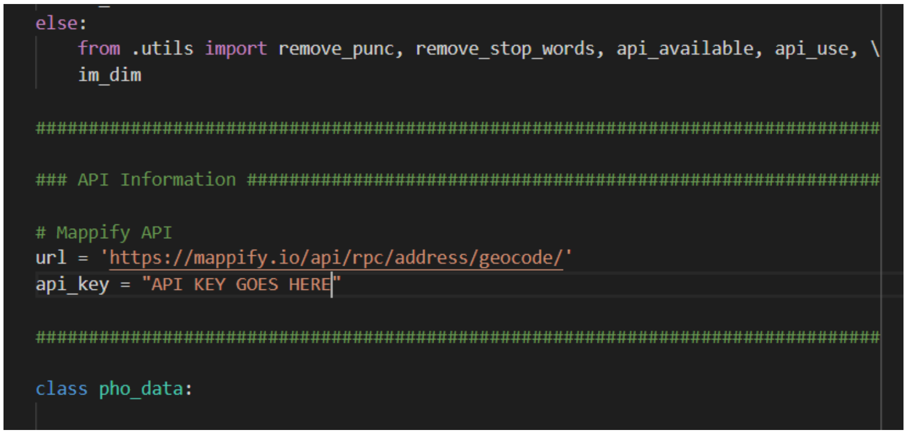
```
python setup.py install
```

This will automatically check for dependencies and install LibClean and missing packages.

**Note:** setup.py must be run from the console that has python in the PATH. If you are using Anaconda [Anaconda, 2020] please use the Anaconda Prompt terminal.

## 2.3 Setting-Up Mappify

For LibClean to successfully geolocate images it must have access to the Mappify API. LibClean does not come with an API key, as the free tier of the Mappify API allows for only 2500 queries per day. In order to use LibClean you must make an account with Mappify (free) and obtain an API key. This key must be then entered into the api_key field located in the "API Information" section of pho_data.py.

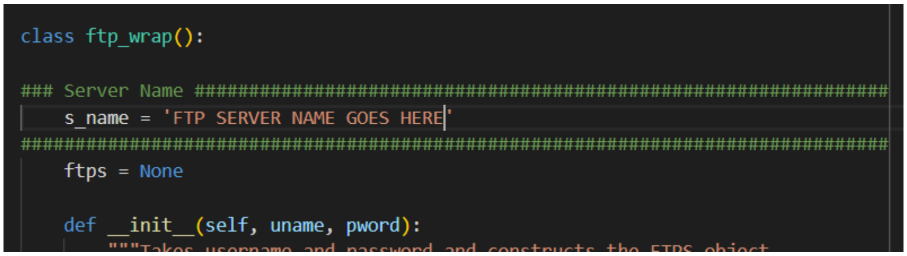


```
else:
    from .utils import remove_punc, remove_stop_words, api_available, api_use, \
    im_dim

################################################################################

### API Information ############################################################

# Mappify API
url = 'https://mappify.io/api/rpc/address/geocode/'
api_key = "API KEY GOES HERE"

################################################################################

class pho_data:
```

Figure 1: The location of the api_key field that must be updated.

## 2.4 Setting-Up Server Access

For LibClean to successfully push images to the website server it must have access via FTP. In order to use this functionality within LibClean, you must enter the server name into the s_name field located in the "Server Name" section of ftp_class.py.



```
class ftp_wrap():

### Server Name ################################################################
    s_name = 'FTP SERVER NAME GOES HERE'
################################################################################
    ftps = None

    def __init__(self, uname, pword):
        """Takes username and password and constructs the FTPS object
```

Figure 2: The location of the s_name field that must be updated.

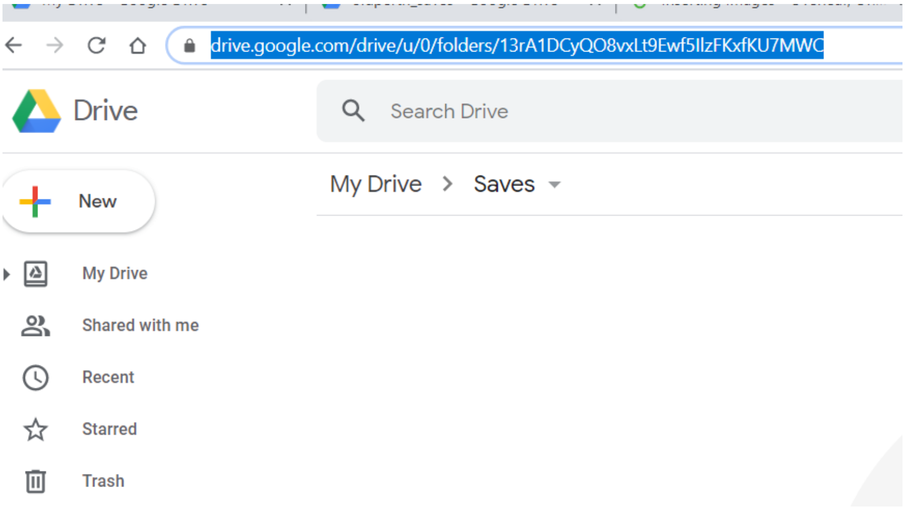
## 2.5 Setting-Up Google Drive

Before LibClean can use a google drive to create persistent catalogue saves, the host drive must be set-up.

1. Choose the google account's you wish LibClean to be given access to and navigate to that account's google drive (https://www.google.com/drive/).

2. In the root directory of your drive create a folder named "saves".

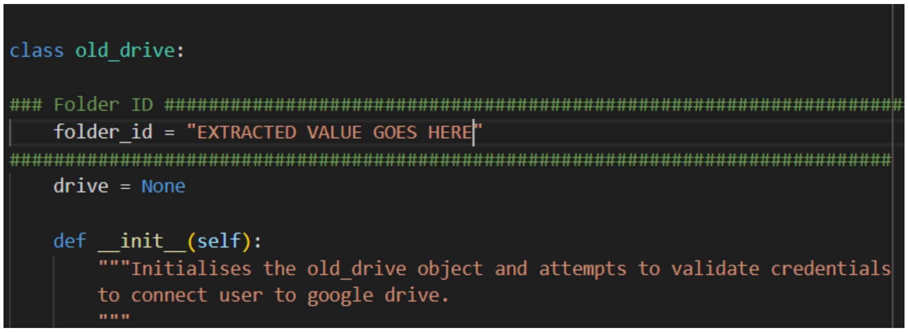
3. Navigate to the newly created saves directory and copy the portion of the url after the last "/".



drive.google.com/drive/u/0/folders/13rA1DCyQO8vxLt9Ewf5IlzFKxfKU7MWC

Figure 3: From the url we would extract, "13rA1DCyQO8vxLt9Ewf5IlzFKxfKU7MWC".

4. Enter the extracted value into the folder_id field located in the "Folder ID" section of old_drive.py.



```
class old_drive:

### Folder ID ####################################################################
    folder_id = "EXTRACTED VALUE GOES HERE"
#################################################################################
    drive = None

    def __init__(self):
        """Initialises the old_drive object and attempts to validate credentials
        to connect user to google drive.
        """
```

Figure 4: The location of the folder_id field that must be updated.

The first time that LibClean attempts to access the google drive a Google authentication flow will open. Simply sign into the account that you wish LibClean to have access to, and when the ”This app isn’t verified page” is displayed, select: advanced>go to Quickstart (unsafe)>allow>allow.

# 3 Functionality

## 3.1 photo_frame.py

The photo_frame class is the outermost layer of LibClean and is where the majority of package functionality is contained.

- **photo_frame.readxml(filename, n=None):** Takes in the name of a file containing a MARC 21 .xml database dump and reads the first n unique photos. If n = None reads in all photos in the database dump. A list of pho_data objects is then created from the parsed image metadata.

- **photo_frame.findLocation():** Iterates through the list of photos and attempts to extract the street address from the parsed metadata.

- **photo_frame.popSearch():** Iterates through all images and searches for a popular place match. If there is a match the relevant fields are updated.

- **photo_frame.manualUpdate(url, num, road, suburb, lat, lon):** Takes 4 strings denoting the image url, street number, street name, suburb and 2 floats denoting the latitude, longitude. If the image exists in the current catalogue updates the relevant fields.

- **photo_frame.saveState():** Saves the current set of photos to a JSON file.

- **photo_frame.loadState(name=’save_state.json’):** Takes the name of the JSON save file and loads a previous object state from the JSON file, merging with the current object state. Duplicate images are discarded.

- **photo_frame.getRecordData(record):** Takes a record object generated from pymarc and returns the author, year and a 2D list containing the image urls and image comments.

- **photo_frame.dataDump():** Dumps current list of images and corresponding metadata into a pipe separated file.

- **photo_frame.genSpreadsheet():** Dumps current list of images and corresponding metadata into a human readable CSV file.

- **photo_frame.geo_locate_images(n=None):** Attempts to geolocate n images. If n is not specified or bigger than the number of images stored, all images are geolocated. An error is thrown if the API is unavailable.

- **photo_frame.get_dims():** Iterates through the stored images, and if the image has a street name their dimensions are extracted from their url.

- **photo_frame.add_update(f_name):** Takes the name of the CSV file located in LibClean/update_data and updates the pho_data objects accordingly.

- **photo_frame.get_tot_im():** Returns the total number of images loaded into LibClean.

- **photo_frame.get_tot_loc():** Returns the total number of images that have been successfully geolocated.

- **photo_frame.get_tot_pro():** Returns the total number of images that have been searched for a street addresses.

- **photo_frame.get_tot_togeo():** Returns the number of images awaiting geolocation.

- **photo_frame.__isPresent():** Determines if the url is present in the current set of images and returns a tuple (boolean, numeric index).

## 3.2 pho_data.py

The pho_data class is the innermost layer of LibClean and contains the metadata and functionality for specific images. The class has a composite relationship with the photo_frame class, such that the photo_frame class ”has” a list of pho_data objects. The pho_data class does not offer any LibClean functionality and as such, should not be interacted with by the user.

- **pho_data.__init__(url, comment, year= None, author=None):** Takes url (string image url), comment (string image comment), year (integer year the image was taken), author (string photographer) and constructs pho_data object.
- **pho_data.__str__():** Returns object state as a string.
- **pho_data.ext_address(suburbs, cityStreets, streetFlags, stopWords, popPlaces):** Takes the paths of the word flags and searches the comment field and attempts to extract the location of the image, updating the suburb road and number fields.
- **pho_data.address_bad():** Setter method that sets latitude and longitude fields to ’NA’ and road, number and suburb fields to None. Should be called if there is not enough information to perform get_coordinates() or if it is known the image has no address.
- **pho_data.get_coordinates():** Attempts to connect to the mappify API and determine the latitude and longitude of the image based on the data stored in the object fields. This method should always be run after the ext_address() method is run. If coordinates are found the lat and lon fields are updated accordingly, otherwise if there is no valid address the pho_data address fields are set to None.
- **pho_data.get_im_dim():** Using the extracted url attempts to determine the dimensions of the image.
- **pho_data.save_dict():** Returns the fields as a dictionary for use in the photo_frame saveState() method.
- **pho_data.load_dict(dict):** Takes a dictionary generated from the save_dict() method and copies the object states except for the url and comment.
- **pho_data.update_address(num, street, sub, lat=None, lon=None):** Takes a string street number, string street name and a string suburb name and updates the relevant fields. If lat and lon are set as None they will be positioned by the get coordinates function.
- **pho_data.pop_search(popPlaces):** Takes a list of popular places and performs a search for popular places in the comment field of all images. This includes located images. If a match is found relevant location data is updated.
- **pho_data.__city_street_sweep(cityStreets):** Takes a list of city streets in Perth and extracts any city addresses present in the comment field. If an address is found the street and number fields are updated.

- **pho_data.__street_sweep(streetFlags, stopWords):** Takes a list of common street flags (road, street,...) and a list of phrasing words to remove from the address (into, the on, ...) and searches for possible addresses within the comment field. The longest address with a street number is then selected and the street and number fields are updated.

- **pho_data.__suburb_sweep(suburbs):** Takes a list of Perth suburbs and if the suburb is not already 'Perth' handles searching for a street address and updates the suburb field.

- **pho_data.__pop_sweep(places):** Takes a list of popular places in Perth and handles searching for the place in the metadata. If a match is found the address data, latitude and longitude fields are updated accordingly.

## 3.3 utils.py

The utils module contains the generic functions that are used by other methods and functions within LibClean. While there is little user functionality present, it is not detrimental to allow users to interact with this module.

- **utils.remove_punc(comment):** Takes a string and removes all punctuation except for ':' and then returns the result.
- **utils.remove_stop_words(line, stopWords):** Takes a string and a list of words to remove and returns the string without those words.
- **utils.api_available():** Reads the api_state.csv file in /save_state and determines whether the mappify api can be used for free. Returning a boolean True:use and False:don't use. If a day has passed, the number of available api uses is reset.
- **utils.api_use(url, payload):** Takes in the url of the api and the payload and performs the query. If the query is successful the number of requests is updated in the file /save_states/save_state.txt and the response is returned as a response object. If the query is unsuccessful a None is returned.
- **utils.im_dim(url):** Takes an image url and attempts to determine the dimensions of the image, if there is an error the function prints and error message and returns None.
- **utils.im_dim(url):** Takes an image url and attempts to determine the dimensions of the image, if there is an error the function prints and error message and returns None.
- **utils.loadFlags(file):** Takes in a filename/path and reads the data stored in the file. If it is a single column of data returns a 1D list, if it is multiple columns of data returns a 2D list. Note the first line is skipped as it assumes these are column labels.
- **utils.loadUpdate(file):** Takes a file name or path, loads an update csv file and returns a dictionary with the relevant fields.

### 3.4 web_utils.py

The web_utils module contains the functions that are used to generate the files required to update the image catalogue of the OldPerth website. The files are stored locally until the ftp_utils module is used to transfer them to the server. Users should interact with the web_utils.web_drop(frame) function to generate the update files.

- **web_utils.latlon_format(photo):** Takes a pho_data object and returns the dictionary format for use by latlon_dump().

- **web_utils.thumb_url_form(url):** Takes the SLWA photo url and returns the thumbnail url. Note this function is heavily influenced by the format of the catalogue and could be easily broken if the Library switches to a different url structure.

- **web_utils.url_form(url):** Takes the SLWA photo url and returns the photo url. Note this function is heavily influenced by the format of the catalogue and could be easily broken if the Library switches to a different url structure.

- **web_utils.get_id(frame):** Takes a photo_frame object and returns a list containing each image's uniquely generated ID.

- **web_utils.id_tbl_dump(frame, im_id):** Generates the translation table used by the OldPerth front-end scripts to determine which image the user is giving feedback on.

- **web_utils.get_latlon_data(frame, im_id):** Takes a photo_frame object and the generated image IDs returns the images sorted by lat lon required for lat_lon_dump.

- **web_utils.id_co_dump(frame, im_id):** Takes a photo_frame object and the generated image IDs and generates the ID to coordinate files for the OldPerth website front-end, located in /front-end/id4-to-location.

- **web_utils.pop_json_dump(frame, im_id):** Takes a photo_frame object and the generated image IDs and generates the popular.json file for the OldPerth website front-end, located /front-end.

- **web_utils.lat_lon_count_dump(latlon_data):** Takes the latlon_data generated by get_latlon_data() and generates the lat-lon-counts.js file required to update the OldPerth website front-end, located in /front-end.

- **web_utils.pop_js_dump(latlon_data):** Takes the latlon_data generated by get_latlon_data() and generates the popular-photos.js file required to update the OldPerth website front-end, located in /front-end/js.

- **web_utils.toloc_js_dump(frame):** Takes the photo_frame object and generates the to_loc.js file required to update the OldPerth website front-end, located in /front-end/oldperth_data.

- **web_utils.web_wipe():** Removes the photo data from the website front-end. Note this does not delete the photo_frame JSON save, only images currently in the website.

- **web_utils.web_drop(frame):** Takes a photo_frame object and updates the local website front-end files accordingly.

## 3.5 ftp_utils.py

The ftp_utils module contains the functions that are used to push the files required to update the image catalogue of the OldPerth website. The files are stored locally until the ftp_utils.server_push() function is used to transfer them to the server. Users should interact with the ftp_utils.update_site() function to push the update files.

- **ftp_utils.update_site():** When called, begins the chain of function calls required to update the OldPerth site.
- **ftp_utils.server_clear(w_ftps):** Takes the ftps wrapper object and removes the files that will conflict with an update to the OldPerth site.
- **ftp_utils.server_push(w_ftps):** Takes the wrapper ftps object and pushes the new version of the website.
- **ftp_utils.dir_rm(ftps, path):** Takes the ftps object and the path of a directory on the server and sequentially deletes files until the directory can be deleted. Note: does not delete other directories.
- **ftp_utils.dir_push(w_ftps, l_path, e_path):** Takes the wrapper ftps object, the local path of the directory and the desired external path and sequentially uploads the directory to the server.
- **ftp_utils.load_file(w_ftps, path, fstrm):** Takes the ftps wrapper, external path of the file and the open file stream for the file to be uploaded, and uploads the file to the server. This function retries connection using exponential back-off.

### 3.6 ftp_class.py

The ftp_class contains the ftp wrapper class that circumvents the FTP_TLS OSerror. The class has exponential back-off built-in to mitigate disconnection errors caused by unstable connections. This class contains no LibClean functionality and as such should not be interacted with by the user.

- **ftp_wrap.__init__(uname, pwrod):** Takes username and password and constructs the FTPS object.
- **ftp_wrap.get_connect():** Uses the FTPS wrapper and constructs an FTPS object.
- **ftp_wrap.con_retry():** If the FTPS object has been disconnected from the server will perform exponential back-off to connect up to a wait time of 40 seconds inclusive.

## 3.7 old_drive.py

The old_drive class contains the LibClean google drive functionality. Users should interact with this class only when retrieving or uploading save state JSON files to the google drive.

- **old_drive.__init__():** Initialises the old_drive object and attempts to validate credentials to connect user to google drive.
- **old_drive.list_saves():** Returns a dictionary of the file ids and dates for each of the saves stored on the google drive.
- **old_drive.pull_curr_save():** Pulls the most recently added save. Note: To load save use photo_frame.loadState(name = 'cloud_save_state.json').
- **old_drive.push_save():** Pushes the current OldPerth save_state.json to the google drive. Note: The save is given the name <seconds since epoch>.json
- **old_drive.pull_save():** Takes a save_state id and pulls the save from the google drive.

# 4 Usage

The LibClean package folder should always be kept in the same directory as the front-end folder if any of web_utils or ftp_utils functionality is desired. In general the following functions, methods and classes should be used by a user.

**photo_frame**

- **photo_frame.readxml(filename, n=None):** Takes in the name of a file containing a MARC 21 .xml database dump and reads the first n unique photos. If n = None reads in all photos in the database dump. A list of pho_data objects is then created from the parsed image metadata.
- **photo_frame.findLocation():** Iterates through the list of photos and attempts to extract the street address from the parsed metadata.
- **photo_frame.popSearch():** Iterates through all images and searches for a popular place match. If there is a match the relevant fields are updated.
- **photo_frame.manualUpdate(url, num, road, suburb, lat, lon):** Takes 4 strings denoting the image url, street number, street name, suburb and 2 floats denoting the latitude, longitude. If the image exists in the current catalogue updates the relevant fields.
- **photo_frame.saveState():** Saves the current set of photos to a JSON file.
- **photo_frame.loadState(name='save_state.json'):** Takes the name of the JSON save file and loads a previous object state from the JSON file, merging with the current object state. Duplicate images are discarded.
- **photo_frame.getRecordData(record):** Takes a record object generated from pymarc and returns the author, year and a 2D list containing the image urls and image comments.
- **photo_frame.dataDump():** Dumps current list of images and corresponding metadata into a pipe separated file.
- **photo_frame.genSpreadsheet():** Dumps current list of images and corresponding metadata into a human readable CSV file.
- **photo_frame.geo_locate_images(n=None):** Attempts to geolocate n images. If n is not specified or bigger than the number of images stored, all images are geolocated. An error is thrown if the API is unavailable.
- **photo_frame.get_dims():** Iterates through the stored images, and if the image has a street name their dimensions are extracted from their url.
- **photo_frame.add_update(f_name):** Takes the name of the CSV file located in LibClean/update_data and updates the pho_data objects accordingly.
- **photo_frame.get_tot_im():** Returns the total number of images loaded into LibClean.

- **photo_frame.get_tot_loc():** Returns the total number of images that have been successfully geolocated.

- **photo_frame.get_tot_pro():** Returns the total number of images that have been searched for a street addresses.

- **photo_frame.get_tot_togeo():** Returns the number of images awaiting geolocation.

- **photo_frame.__isPresent():** Determines if the url is present in the current set of images and returns a tuple (boolean, numeric index).

**web_utils**

- **web_utils.web_drop(frame):** Takes a photo_frame object and updates the local website front-end files accordingly.

**ftp_utils**

- **ftp_utils.update_site():** When called, begins the chain of function calls required to update the OldPerth site.

**ftp_utils**

- **ftp_utils.update_site():** When called, begins the chain of function calls required to update the OldPerth site.

**old_drive**

- **old_drive.__init__():** Initialises the old_drive object and attempts to validate credentials to connect user to google drive.

- **old_drive.list_saves():** Returns a dictionary of the file ids and dates for each of the saves stored on the google drive.

- **old_drive.pull_curr_save():** Pulls the most recently added save. Note: To load save use photo_frame.loadState(name = 'cloud_save_state.json').

- **old_drive.push_save():** Pushes the current OldPerth save_state.json to the google drive. Note: The save is given the name <seconds since epoch>.json

- **old_drive.pull_save():** Takes a save_state id and pulls the save from the google drive.

# 5 Input and Output Data

## 5.1 MARC Data

MARC XML files should be stored in the LibClean/marc_data directory before being read into LibClean.

## 5.2 Save States

LibClean catalogue save states are stored locally in the LibClean/save_states directory, while cloud saves are stored on the host google drive. The save states are stored as JSON files and should not be edited by the user.

## 5.3 Update CSV Files

Update CSV files should be stored in LibClean/update_data before being parsed into LibClean. The OldPerth website has links to google forms which are used to generate the update files. However, LibClean is able to parse any update CSV file provided it is a tabular CSV with the following column names: url,Street Number,Street/Road Name,Suburb.

## 5.4 Data Dumps

All data dumps generated by LibClean are stored in the LibClean/data_dump directory.

# References

[Anaconda, 2020] Anaconda (2020). *Anaconda Software Distribution.*

[Python Core Team, 2020] Python Core Team (2020). *Python: A dynamic, open source programming language.*

[Summers, 2019] Summers, E. (2019). *pymarc.*