Tagger API (beta)

==========

Authors: Michael Koutroumpas <m.koutroumpas@ed.ac.uk>;
James Reid <james.reid@ed.ac.uk>
Version: v1.5
Date: Aug 2012

Preamble

Tagger is an application and web service developed by EDINA under funding from JISC as part of the latters Discovery Programme. Tagger's purpose is to provide tools for 3rd parties to use in order to enrich and expose open metadata – in this case, specifically metadata embedded within digital assets such as images. Tagger provides a means to expose and manipulate these 'hidden metadata' in support of Open Metadata principles. The Tagger database has been seeded with sample images (c.0.25M) from Geograph[1]. Additional user uploaded files have been added to the corpus and will accrue as the service matures.

The Application Programming Interface (API) described here relates to the web service "Tagger" available at endpoint:

tagger.edina.ac.uk/ws/

The API is purely REST based using HTTP POST/GET requests. A "unique token" called *uuid* is used for session tracking. This uuid is created with either the **uploadfile** or **nearest/edit** methods, or by sharing the known uuid(s) (permalinks) from previous sessions.

The output is *always* application/json unless it is the actual media file resource being downloaded. All examples have been tested with the curl command line application as well as Firefox's AJAX XHR client through Javascript.

Usage

Initially, a media file (typically an image) must be uploaded in order to obtain a **uuid** associated with the resource. This uuid can later be used to perform all other permissible operations such as:

- *Getting all probed metadata associated with the uploaded file*

.

- *Changing existing metadata.*
- *Export metadata as a XMP[2] sidecar file.*
- *Adding or Deleting metadata.*
- *Add/Purge/Fuzzify geolocation recorded in EXIF metadata.*
- *Finding nearest images*
- *Loading (acquiring uuid(s)) of nearestimages.*
- *Uploading images to Dropbox.*

NOTE: Always use the **uploadfile** or **nearest/edit** calls first in order to get a **uuid** for use in subsequent API calls.

1 http://www.geograph.org.uk/

2 http://en.wikipedia.org/wiki/Sidecar_file

The design of the API calls follow the following simple rules:

- Functions are part of the url
- Mandatory arguments are also part of the URL (if possible)
- optional arguments are sent with either GET or POST depending on their size.

e.g. a function myfunct(mandatory_parm, optional_parm) would be called as follows:

http://host/ws/myfunct/mandatory_parm_value?optional_parm=value

API Calls

Overview

Below is an alphabetical listing of all Tagger API functions for release 1.5:

API Call	Brief description
addKeyword(uuid, key)	Add a "keyword" to a media file. Keywords are similar to tags in Flickr.
anonymise(uuid)	Delete all GPS related metadata from the media file associated with *uuid*.
edit(id)	Edit metadata for target resource *id*.This command is used after the nearest command. The original file will not be edited - a fresh copy of the image will be created with its own *uuid*.
export(uuid)	Download all the metadata for a media file associated with uuid as an *XMP* sidecar file.
geotag(uuid,lon,lat)	High-level API call used for *geotagging*.
geotag(uuid,placename)	Alternate call used for *lazy geotagging*. Does implicit remote lookup of Unlock global gazetteer API [unlock.edina.ac.uk].
getfile(uuid)	Download the media file associated with *uuid*.
getKeywords(uuid)	Request all "keywords" associated with a media file returned as a CSV.
getLicense(uuid)	Return the license associated with a media file.
getUUIDs(bbox,key)	Returns all *uuids* for user uploaded and Geograph media files within a spatial extent and/or files having a keyword.
loadMetadata(uuid, group=ALL)	Get all probed properties of media file described by *uuid* and return them as a JSON object
nearest(lon,lat)	Returns a JSON list of up to 10 images in a 2km distance from the specified lon/lat point.
saveMetadata(uuid,new_metadata,group)	Get all probed properties of mediafile described by *uuid* and return them as a JSON object
setLicense(uuid, license)	Sets the license of a media file.
uploadfile	Upload a file and return a *uuid* as handle for further calls.

Details

Each of the above API calls are detailed with examples below.

uploadfile

A media file is POSTed using the Content-Type multipart/form-data according to the RFC 2388. Upon success, a uuid is returned which should subsequently be used for all further API calls:

1. Use an HTTP client to POST a file to tagger.edina.ac.uk/ws/uploadfile

2. Return call is a JSON object:

    { uuid:<the uuid>, error:0, msg:<error message> } on success or

    { uuid:0, error: ERROR_CODE } on failure

error codes are:

    0: success (uuid is valid)

    1: unsupported media file

    2: internal error

Example with curl: -- Posting a file named MyMedia.jpg in the current directory:

curl --form fileInput=@MyMedia.jpg tagger.edina.ac.uk/ws/uploadFile

returns:

{"msg": "Success!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

loadMetadata(uuid, group=ALL)

Get all probed properties of mediafile described by uuid and return them as a JSON object. Optionally, the group parameter may be used to specify the target metadata group (e.g. EXIF, Custom, XMP etc.). This should normally be used for writing Custom metadata group for data that are stored and indexed in the Tagger database.

Example with curl:

curl tagger.edina.ac.uk/ws/loadMetadata/63c6fa485ac86a02f07eb5fee2fdd518

Returns:

{"FileName": "MyMedia.jpg", "FileType": "JPEG", "MIMEType": "image/jpeg",
"YResolution": 180, "ResolutionUnit": 2, "FocalPlaneYSize": 15.494, ... }

saveMetadata(uuid,new_metadata,group)

A POST request of a JSON object containing the updated metadata fields. Only the changed values should be submitted. To delete a tag, just use an empty string as a value.

Example with curl : - to create or update a tag named Author with value John Doe and to delete a tag named Copyright in one operation:

curl --data 'metadata={"Author":"John Doe","Copyright":""}' \

tagger.edina.ac.uk/ws/saveMetadata/63c6fa485ac86a02f07eb5fee2fdd518

Optionally, the group parameter may be used to specify the target metadata group (e.g. EXIF, Custom, XMP etc.). This should normally be used for writing Custom metadata that are stored and indexed in the Tagger database instead of being hardcoded in the file. For example the above operation would write the metadata in the database by appending using the extra POST parameter group=Custom:

curl --data 'metadata={"Author":"John Doe","Copyright":""}&group=Custom' \
tagger.edina.ac.uk/ws/saveMetadata/63c6fa485ac86a02f07eb5fee2fdd518

Returns:

{"msg": "Metadata saved!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

geotag(uuid,lon,lat)

High-level API call used for geotagging. It is equivalent to calling saveMetadata to save the appropriate GPS parameters.

Example with curl:

curl tagger.edina.ac.uk/ws/geotag/63c6fa485ac86a02f07eb5fee2fdd518?lon=55.9545&lat=-3.1901

Returns:

{"msg": "Metadata saved!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

geotag(uuid,placename)

For convenience we provide an alternate function which allows for lazy geotagging. Simply providing a textual placename e.g. 'Edinburgh' will result in attempts to do a lookup (using the EDINA Unlock Places global gazetteer API), which will embed the GPS coordinate information returned from Unlock into the resource. Unlock attempts to rank results if there are multiple response candidates that meet search criteria e.g. Edinburgh will return candidates for c.120 places named 'Edinburgh' globally. Tagger in this release of the API will simply choose the *first returned candidate*. In most cases this will be sufficient, however, if an alternative candidate is required, users are advised to use the Unlock API directly to identify the appropriate lat/long and to then call Tagger with **geotag(uuid,lon,lat).** Full details of the Unlock API are available at:

http://unlock.edina.ac.uk/places/introduction

Example with curl:

curl tagger.edina.ac.uk/ws/geotag/6563cdd722e647dd87559e969a26c4ae?placename=edinburgh

Returns:

{"msg": "Metadata saved!", "uuid": "6563cdd722e647dd87559e969a26c4ae", "error": 0}

addKeyword(uuid, key)

A GET request for associating a "keyword" with a media file. Keywords are similar to tags in Flickr.

Example with curl:

curl tagger.edina.ac.uk/ws/addKeyword/63c6fa485ac86a02f07eb5fee2fdd518?key=water

Returns:

{"msg": "Keyword saved!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

getKeywords(uuid)

A GET request for all "keywords" associated with a media file returned as a CSV.

Example with curl:

curl tagger.edina.ac.uk/ws/getKeywords/63c6fa485ac86a02f07eb5fee2fdd518

Returns:

{ "keywords" : "river,boat", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

getUUIDs(bbox,key)

Returns all uuids for user uploaded and existing (Geograph) media files within a bounding box (bbox=xmin,ymin,xmax,ymax) and/or files having a keyword "some keyword".

Example with curl:

curl 'tagger.edina.ac.uk/ws/getUUIDs?bbox=-3.2650,55.5930,-3.2620,55.5950'

returns:

{"tagger_uuids": "b2b259d749c8c467bb4ebaad513310d0", "geograph_uuids": "145927,1874597", "error": 0}

or with a keyword search: curl 'tagger.edina.ac.uk/ws/getUUIDs?key=testtag'

setLicense(uuid, license)

Sets the license of a media file. The license can be either a Creative Commons compatible license which will be stored as an XMP tag or an arbitrary string (e.g. a URL to a full legal document) which will be stored as a custom property.

Example with curl:

curl tagger.edina.ac.uk/ws/setLicense/63c6fa485ac86a02f07eb5fee2fdd518?license=CC-SA

Returns:

{"msg": "License saved!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

getLicense(uuid)

Returns the license associated with a media file.

Example with curl:

curl tagger.edina.ac.uk/ws/getLicense/63c6fa485ac86a02f07eb5fee2fdd518

Returns:

{"Licensed": "CC-SA", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

getfile(uuid)

Download the file associated with uuid. The file should have all new metadata hardcoded from previous API calls to saveMetadata:

Example with curl:

curl -o mymedia.jpg tagger.edina.ac.uk/ws/getfile/63c6fa485ac86a02f07eb5fee2fdd518

Returns:
The actual binary file with its MIME type set according to the media file. This url can also be used directly in the browser or as a static link.

anonymise(uuid)

A GET request to delete all GPS related metadata of the media file associated with uuid.

Example with curl:

curl tagger.edina.ac.uk/ws/anonymise/63c6fa485ac86a02f07eb5fee2fdd518

Returns:

{"msg": "Success!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

export(uuid)

A GET request to download all metadata of the media file associated with uuid as an XMP sidecar file.

Example with curl:

curl -o mymedia.xmp tagger.edina.ac.uk/ws/export/63c6fa485ac86a02f07eb5fee2fdd518

Returns: the XMP sidecar file as XML.

nearest(lon,lat)

Searches the database for all Creative-Commons licensed user-uploaded data and/or Geograph images around the specified point. It returns a JSON list of up to 10 images in a 2km distance from the specified lon/lat point.

Example with curl : - to search database for images around point -3.598728,55.731181:

curl 'tagger.edina.ac.uk/ws/nearest?lon=-3.598728&lat=55.731181'

Returns: a JSON array of nearest images (along with their IDs). The fields of the array are :

[id, title, short description, tags, long description, lon,lat, distance]

For example the above request returns: (only two rows displayed)

[[160789, "Callum Black", "Craigiehall", null, "Farm on the pleasant minor road to Wester Yardhouses.", -3.5987279999999999, 55.731180999999999, 0.0],

[160791, "Callum Black", "Greenshieldhouse", null, "Cluster of farm buildings at the start of the road to Wester Yardhouses.", -3.6050939999999998, 55.731098000000003, 400.05211705678636]]

edit(id)

This command is used after the **nearest** command. The original file will not be edited - a fresh copy of the image will be created with its own uuid.

Example with curl: - if we want to start editing the metadata of the above example "Craigiehall" with id 160789 we would execute:

curl tagger.edina.ac.uk/ws/edit/160789

Returns: a JSON object exactly like, uploadfile:

{ uuid:<the uuid>, error:0, msg:<error message> } on success
or
{ uuid:0, error: ERROR_CODE } on failure.

Dropbox API

Quick Tutorial:

This is a cookbook style tutorial to quickly get an impression of the dropbox API. Let's assume for example that a mobile app tries to use the Tagger API to:

1. upload an image

2. geotag it with known coordinates- typically taken from the location aware device itself at time of image capture

3. Upload it to the Dropbox account of the user of the mobile app

In our case the mobile app is just curl as in all the examples so far. The set of commands (assuming a linux like terminal environment), are as follows:

bash$ curl --form fileInput=@MyMedia.jpg tagger.edina.ac.uk/ws/uploadFile

{"msg": "Success!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

bash$ curl tagger.edina.ac.uk/ws/geotag/63c6fa485ac86a02f07eb5fee2fdd518?lon=55.9545&lat=-3.1901

{"msg": "Success!", "uuid": "63c6fa485ac86a02f07eb5fee2fdd518", "error": 0}

bash$ curl tagger.edina.ac.uk/ws/dropbox_login

{"url": "https://www.dropbox.com/1/oauth/authorize?oauth_token=tchzufdt438y76q",

"state": 0, "req_key": "tchzufdt438y76q"}

At this point the user needs to open the url :

https://www.dropbox.com/1/oauth/authorize?oauth_token=tchzufdt438y76q

inside a browser and let the mobile up know when the process is complete (e.g. by hitting a button). This can be made easier if the mobile app has an embedded browser.

bash$ curl tagger.edina.ac.uk/ws/dropbox_callback/tchzufdt438y76q

{"error": 0, "msg": "tchzufdt438y76q"}

Now the mobile app will not have to relogin again by saving the request key "tchzufdt438y76" for future API calls.

bash$ curl
tagger.edina.ac.uk/ws/dropbox_upload/63c6fa485ac86a02f07eb5fee2fdd518/tchzufdt438y76q

{"error": 0, "msg": "File Uploaded", "url":
"https://dl.dropbox.com/0/view/wiz2e7pct242oer/Apps/Tagger/myfile.jpg",

"expires": "Wed, 27 Jun 2012 14:34:27 0000", "error": 0}

NOTE: Another option would be to call the dropbox_login function with the optional callback argument that would make the whole process more seamless. In that case, though, the mobile app would need to start polling the Tagger login function until it receives a response with state = 1 (already logged in).

dropbox_login(req_key, callback)
Initiate a dropbox conection.

Arguments:
req_key (optional); This is the request key used in a previous connection. It can be passed here to resume that connection without having to relogin.

callback (optional); A URL pointing to a redirection endpoint for dropbox.

Returns:
A JSON object with a url property that the client can use to give access to the app, a request key req_key that is used by Tagger to keep track of the connection and a state variable which can be either 0 (token needs verification using url) or 1 (already connected, you can start issuing commands).

Example with curl:

curl tagger.edina.ac.uk/ws/dropbox_login

Returns: a JSON object similar to:

{"url": "https://www.dropbox.com/1/oauth/authorize?oauth_token=tchzufdt438y76q",
"state": 0, "req_key": "tchzufdt438y76q"}

dropbox_callback(req_key)

Callback to indicate that verification was done by user. User MUST provide the req_key argument for session tracking.

Arguments:
req_key (optional); This is the request key given to the API user by the dropbox_login,dropbox_login function.

Returns:

A JSON object with an error and a msg attribute. If error is 0 then the operation was successful and the msg is set to the req_key that was passed to the function.

Example with curl:

curl tagger.edina.ac.uk/ws/dropbox_callback/tchzufdt438y76q

Returns: a JSON object similar to:

{"error": 0, "msg": "tchzufdt438y76q"}

dropbox_upload(uuid, req_key)

Upload command.

Arguments:
uuid; The uuid of the media file to upload.

req_key; This is the request key given to the API user by the dropbox_login, dropbox_login function.

Returns:
A JSON object with an error and a msg property. If there was an error then the msg will be the exact error message as passed to Tagger by the Dropbox web service. On success two extra properties are available: url which is a direct link to the file in dropbox and expires which is the date when the url stops being valid (in about a month after creation).

Example with curl:

curl tagger.edina.ac.uk/ws/dropbox_upload/ddfd47912fe45771b2e93b9d11624182/tchzufdt438y76q

Returns: a JSON object similar to:

{"error": 0, "msg": "File Uploaded", "url":
"https://dl.dropbox.com/0/view/wiz2e7pct242oer/Apps/Tagger/myfile.jpg",
"expires": "Wed, 27 Jun 2012 14:34:27 0000", "error": 0}

Caveat Emptor

While a whole slew of different media files can be probed for their metadata, not all can be modified or written (most notably GeoTiff). Tagger uses the exiftool at its core for metadata processing and one should therefore look at the upstream documentation at:

http://www.sno.phy.queensu.ca/~phil/exiftool/exiftool_pod.html

for a list of supported tags and file types.

Custom data, which are not hardcoded in the media file, may be arbitrary and can be manipulated and returned at will.