PyOpenload Documentation

Release 0.4

Mohaned Magdy

Aug 19, 2017

Contents:

CHAPTER 1

Overview

PyOpenload is a python wrapper for Openload.co API.

With PyOpenload you can manage your `openload.co` account (Access account info, Upload files, ...) with no need to understand how to use Openload.co API.

Learn more about what you can do at Openload.co API.

License

About

PyOpenload is maintained by:

- Mohaned Magdy

CHAPTER 2

Download

Package

Source: https://pypi.org/project/pyopenload/ or https://pypi.python.org/pypi/pyopenload/

Github: https://github.com/mohan3d/PyOpenload/

Documentation

PDF: https://media.readthedocs.org/pdf/pyopenload/latest/pyopenload.pdf

HTML: https://media.readthedocs.org/htmlzip/pyopenload/latest/pyopenload.zip

CHAPTER 3

Installation

Quick install

Get PyOpenload from the Python Package Index.

or install it with pip

```
pip install PyOpenload
```

You can install the development version (at github.com) with

```
pip install git+https://github.com/mohan3d/PyOpenload
```

Installation from source

Github

- Clone the PyOpenload repository

```
git clone https://github.com/mohan3d/PyOpenload.git
```

- Change directory to `PyOpenload`
- Run `python setup.py install`

Requirements

Requests

requests is the only required dependency.

CHAPTER 4

Reference

class openload.**OpenLoad**(*api_login*, *api_key*)

classmethod _check_status(*response_json*)
Check the status of the incoming response, raise exception if status is not 200.

Parameters **response_json** (*dict*) – results of the response of the GET request.

Returns None

classmethod _process_response(*response_json*)
Check the incoming response, raise error if it's needed otherwise return the incoming response_json

Parameters **response_json** (*dict*) – results of the response of the GET request.

Returns results of the response of the GET request.

Return type dict

_get(*url*, *params=None*)
Used by every other method, it makes a GET request with the given params.

Parameters

- **url** (*str*) – relative path of a specific service (account_info, ...).
- **params** (dict, optional) – contains parameters to be sent in the GET request.

Returns results of the response of the GET request.

Return type dict

account_info()
Requests everything account related (total used storage, reward, ...).

Returns

dictionary containing account related info.

```
{
  "extid": "extuserid",
  "email": "jeff@openload.io",
  "signup_at": "2015-01-09 23:59:54",
  "storage_left": -1,
  "storage_used": "32922117680",
  "traffic": {
    "left": -1,
    "used_24h": 0
  },
  "balance": 0
}
```

Return type dict

prepare_download(*file_id*)

Makes a request to prepare for file download, this download preparation will be used before get_download_link method.

Parameters **file_id** (*str*) – id of the file to be downloaded.

Returns

dictionary containing (ticket, captcha info, ...).

```
 {
   "ticket": "72fA-_Lq8Ak~~1440353112~n~~0~nXtN3RI-nsEa28Iq",
   "captcha_url": "https://openload.co/dlcaptcha/b92eY_nfjV4.png",
   "captcha_w": 140,
   "captcha_h": 70,
   "wait_time": 10,
   "valid_until": "2015-08-23 18:20:13"
}
```

Return type dict

get_download_link(*file_id*, *ticket*, *captcha_response=None*)

Requests direct download link for requested file, this method makes use of the response of prepare_download, prepare_download must be called first.

Parameters

- **file_id** (*str*) – id of the file to be downloaded.
- **ticket** (*str*) – preparation ticket is found in prepare_download response, this is why we need to call prepare_download before get_download_link.
- **captcha_response** (str, optional) – sometimes prepare_download will have captcha url to be solved, first, this is the solution of the captcha.

Returns

dictionary containing (file info, download url, ...).

```
{
  "name": "The quick brown fox.txt",
  "size": 12345,
  "sha1": "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12",
  "content_type": "plain/text",
  "upload_at": "2011-01-26 13:33:37",
  "url": "https://abvzps.example.com/dl/l/4spxX_-cSO4/
↪The+quick+brown+fox.txt",
```

```
  "token": "4spxX_-cSO4"
}
```

Return type dict

file_info (*file_id*)
Used to request info for a specific file, info like size, name, ...

Parameters **file_id** (*str*) – File-ID(s), single file or comma-separated (max. 50)

Returns

dictionary containing file(s) info, each key represents a file_id.

```
{
   "72fA-_Lq8Ak3": {
      "id": "72fA-_Lq8Ak3",
      "status": 200,
      "name": "The quick brown fox.txt",
      "size": 123456789012,
      "sha1": "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12",
      "content_type": "plain/text",
   },
   "72fA-_Lq8Ak4": {
      "id": "72fA-_Lq8Ak4",
      "status": 500,
      "name": "The quick brown fox.txt",
      "size": false,
      "sha1": "2fd4e1c67a2d28fced849ee1bb76e7391b93eb12",
      "content_type": "plain/text",
   },
   ...
 }
```

Return type dict

upload_link (*folder_id=None*, *sha1=None*, *httponly=False*)
Makes a request to prepare for file upload.

Note: If folder_id is not provided, it will make and upload link to the `Home` folder.

Parameters

- **folder_id** (`str`, optional) – folder-ID to upload to.
- **sha1** (`str`, optional) – expected sha1 If sha1 of uploaded file doesn't match this value, upload fails.
- **httponly** (`bool`, optional) – If this is set to true, use only http upload links.

Returns

dictionary containing (url: will be used in actual upload, valid_until).

```
{
    "url": "https://1fiafqj.oloadcdn.net/uls/nZ8H3X9e0AotInbU",
    "valid_until": "2017-08-19 19:06:46"
}
```

Return type dict

upload_file(*file_path*, *folder_id=None*, *sha1=None*, *httponly=False*)
Calls upload_link request to get valid url, then it makes a post request with given file to be uploaded. No need to call upload_link explicitly since upload_file calls it.

Note: If folder_id is not provided, the file will be uploaded to `Home` folder.

Parameters

- **file_path** (*str*) – full path of the file to be uploaded.
- **folder_id** (`str`, optional) – folder-ID to upload to.
- **sha1** (`str`, optional) – expected sha1 If sha1 of uploaded file doesn't match this value, upload fails.
- **httponly** (`bool`, optional) – If this is set to true, use only http upload links.

Returns

dictionary containing uploaded file info.

```
{
    "content_type": "application/zip",
    "id": "0yiQTPzi4Y4",
    "name": 'favicons.zip',
    "sha1": 'f2cb05663563ec1b7e75dbcd5b96d523cb78d80c',
    "size": '24160',
    "url": 'https://openload.co/f/0yiQTPzi4Y4/favicons.zip'
 }
```

Return type dict

remote_upload(*remote_url*, *folder_id=None*, *headers=None*)
Used to make a remote file upload to openload.co

Note: If folder_id is not provided, the file will be uploaded to `Home` folder.

Parameters

- **remote_url** (*str*) – direct link of file to be remotely downloaded.
- **folder_id** (`str`, optional) – folder-ID to upload to.
- **headers** (`dict`, optional) – additional HTTP headers (e.g. Cookies or HTTP Basic-Auth)

Returns

dictionary containing ("id": uploaded file id, "folderid").

```
{
    "id": "12",
    "folderid": "4248"
}
```

Return type dict

remote_upload_status (*limit=None*, *remote_upload_id=None*)
Checks a remote file upload to status.

Parameters

- **limit** (int, optional) – Maximum number of results (Default: 5, Maximum: 100).
- **remote_upload_id** (str, optional) – Remote Upload ID.

Returns

dictionary containing all remote uploads, each dictionary element is a dictionary.

```
{
    "24": {
      "id": "24",
      "remoteurl": "http://proof.ovh.net/files/100Mio.dat",
      "status": "new",
      "folderid": "4248",
      "added": "2015-02-21 09:20:26",
      "last_update": "2015-02-21 09:20:26",
      "extid": False,
      "url": False
    },
    "22": {
      "id": "22",
      "remoteurl": "http://proof.ovh.net/files/1Gio.dat",
      "status": "downloading",
      "bytes_loaded": "823997062",
      "bytes_total": "1073741824",
      "folderid": "4248",
      "added": "2015-02-21 09:20:26",
      "last_update": "2015-02-21 09:21:56",
      "extid": False,
      "url": False
    },
    ...
}
```

Return type dict

list_folder (*folder_id=None*)
Request a list of files and folders in specified folder.

Note: if folder_id is not provided, Home folder will be listed

Parameters **folder_id** (str, optional) – id of the folder to be listed.

Returns

dictionary containing only two keys ("folders", "files"), each key represents a list of dictionaries.

```
{
  "folders": [
    {
      "id": "5144",
      "name": ".videothumb"
    },
```

```
    {
      "id": "5792",
      "name": ".subtitles"
    },
    ...
  ],
  "files": [
    {
      "name": "big_buck_bunny.mp4.mp4",
      "sha1": "c6531f5ce9669d6547023d92aea4805b7c45d133",
      "folderid": "4258",
      "upload_at": "1419791256",
      "status": "active",
      "size": "5114011",
      "content_type": "video/mp4",
      "download_count": "48",
      "cstatus": "ok",
      "link": "https://openload.co/f/UPPjeAk--30/big_buck_bunny.mp4.
↪mp4",
      "linkextid": "UPPjeAk--30"
    },
    ...
  ]
}
```


Return type dict

rename_folder (*folder_id*, *name*)
Sets a new name for a folders

Note: folder_id(s) can be found in list_folder return.

Parameters

- **folder_id** (*str*) – id of the folder to be renamed.
- **name** (*str*) – new name for the provided folder.

Returns True if folder is renamed, otherwise False.

Return type bool

rename_file (*file_id*, *name*)
Sets a new name for a file

Parameters

- **file_id** (*str*) – id of the file to be renamed.
- **name** (*str*) – new name for the provided file.

Returns True if file is renamed, otherwise False.

Return type bool

delete_file (*file_id*)
Removes one of your files

Parameters **file_id** (*str*) – id of the file to be deleted.

Returns True if file is deleted, otherwise False.

Return type bool

convert_file(*file_id*)
Converts previously uploaded files to a browser-streamable format (mp4 / h.264).

Parameters **file_id** (*str*) – id of the file to be converted.

Returns True if conversion started, otherwise False.

Return type bool

running_conversions(*folder_id=None*)
Shows running file converts by folder

Note: If folder_id is not provided, `Home` folder will be used.

Parameters **folder_id** (`str`, optional) – id of the folder to list conversions of files exist in it.

Returns

list of dictionaries, each dictionary represents a file conversion info.

```
[
  {
    "name": "Geysir.AVI",
    "id": "3565411",
    "status": "pending",
    "last_update": "2015-08-23 19:41:40",
    "progress": 0.32,
    "retries": "0",
    "link": "https://openload.co/f/f02JFG293J8/Geysir.AVI",
    "linkextid": "f02JFG293J8"
  },
  ....
]
```

Return type list

failed_conversions()
Not yet implemented, openload.co said "Coming soon ...".

Raises `NotImplementedError`

splash_image(*file_id*)
Shows the video splash image (thumbnail)

Parameters **file_id** (*str*) – id of the target file.

Returns url for the splash image.

Return type str

CHAPTER 5

Examples

`username` and `key` can be found in openload user settings.

Account

Account Infos

Get everything account related (total used storage, reward, ...).

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

openload = OpenLoad(username, key)
info = openload.account_info()

print(info)
```

Download

Download Ticket

Generate a download token, will be used to generate direct download link.

```
from __future__ import print_function

from openload import OpenLoad
```

```
username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file will be downloaded'

openload = OpenLoad(username, key)
resp = openload.prepare_download(file_id)

ticket = resp.get('ticket')
captcha_url = resp.get('captcha_url')

print(ticket)
print(captcha_url)
```

Download Link

Generate a download link, after generating a download ticket.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file will be downloaded'
ticket = 'Ticket found in `prepare_download` response'
captcha_response = 'Solution of captcha found in `prepare_download` response'

openload = OpenLoad(username, key)
resp = openload.get_download_link(file_id,
                                  ticket,
                                  captcha_response)

direct_download_url = resp.get('url')

print(direct_download_url)
```

Full example

1. Generate a download token.
2. Solve captcha if needed.
3. Generate direct download url.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file will be downloaded'

openload = OpenLoad(username, key)
```

```
# Get a download ticket and captcha url.
preparation_resp = openload.prepare_download(file_id)
ticket = preparation_resp.get('ticket')

# Sometimes no captcha is sent in openload.co API response.
captcha_url = preparation_resp.get('captcha_url')

if captcha_url:
    # Solve captcha.
    captcha_response = solve_captcha(captcha_url)
else:
    captcha_response = ''

download_resp = openload.get_download_link(file_id, ticket, captcha_response)
direct_download_url = download_resp.get('url')

# Process download url.
print(direct_download_url)
```

File Info

Check the status of a file (id, status, name, size, sha1, content_type).

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file(s) to be checked'

openload = OpenLoad(username, key)
info = openload.file_info(file_id)

# Process info.
print(info)
```

Upload

Get an Upload URL

You may need to use this method only if you want to re-implement `upload_file` in a different way.

Generate upload url, will be used to upload a file.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

openload = OpenLoad(username, key)
resp = openload.upload_link()
```

```
upload_link = resp.get('url')

print(upload_link)
```

Upload File

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

openload = OpenLoad(username, key)
resp = openload.upload_link()
upload_link = resp.get('url')

print(upload_link)
```

Remote Upload

Add Remote Upload

Upload lastest pyopenload documentation pdf.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

pdf_url = 'https://media.readthedocs.org/pdf/pyopenload/latest/pyopenload.pdf'

openload = OpenLoad(username, key)
resp = openload.remote_upload(pdf_url)

file_id = resp.get('id')

print(file_id)
```

Check Remote Upload Status

Check the status of queued remote uploads.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
```

```
openload = OpenLoad(username, key)
resp = openload.remote_upload_status()

print(resp)
```

File/Folder Management

List Folder

List `Home` (The main directory).

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

openload = OpenLoad(username, key)
resp = openload.list_folder()

print(resp)
```

Rename Folder

Rename a specific folder.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
folder_id = 'Id of the folder will be renamed'

openload = OpenLoad(username, key)
resp = openload.rename_folder(folder_id, '<NEW NAME>')

print(resp)
```

Rename File

Rename a specific file.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
```

```
file_id = 'Id of the file will be renamed'

openload = OpenLoad(username, key)
resp = openload.rename_file(file_id, '<NEW NAME>')

print(resp)
```

Delete File

Delete a specific file.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file will be deleted'

openload = OpenLoad(username, key)
resp = openload.delete_file(file_id)

print(resp)
```

Converting files

Convert a file

Convert previously uploaded file to a browser-streamable format `mp4 / h.264`

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file(s) to be checked'

openload = OpenLoad(username, key)
resp = openload.convert_file(file_id)

print(resp)
```

Show running file converts

List all running conversions.

```
from __future__ import print_function

from openload import OpenLoad

```

```
username = 'FTP Username/API Login'
key = 'FTP Password/API Key'

openload = OpenLoad(username, key)
resp = openload.running_conversions()

print(resp)
```

Show failed file converts

Coming soon ... (Not yet implemented by openload.co API).

Get splash image

Get a download url of splash image for a specific uploaded file.

```
from __future__ import print_function

from openload import OpenLoad

username = 'FTP Username/API Login'
key = 'FTP Password/API Key'
file_id = 'Id of the file will be downloaded'

openload = OpenLoad(username, key)
resp = openload.splash_image(file_id)

print(resp)
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index