instagram_private_api_extensions Documentation

Release 0.3.9

ping

Nov 14, 2019

Usage

An extension module to instagram_private_api to help with common tasks such as posting a photo or video.

CHAPTER 1

Features

1. *Media*: Edits a photo/video so that it complies with Instagram's requirements by:
    - Resizing
    - Cropping to fit the minimum/maximum aspect ratio
    - Generating the video thumbnail image
    - Clipping the video duration if it is too long
    - Changing the format/encoding
2. *Pagination*: Page through an api call such as `api.user_feed()`.
3. *Live*: Download an ongoing IG live stream. Requires ffmpeg installed.
4. *Replay*: Download an IG live replay stream. Requires ffmpeg installed.

1.1 Installation

1.1.1 Pip

Install via pip

```
$ pip install git+ssh://git@github.com/ping/instagram_private_api_extensions.git@0.3.9
```

Update your install with the latest release

```
$ pip install git+ssh://git@github.com/ping/instagram_private_api_extensions.git@0.3.
↪9 --upgrade
```

Force an update from source

```
$ pip install git+ssh://git@github.com/ping/instagram_private_api_extensions.git --
↪upgrade --force-reinstall
```

1.1.2 Source Code

The library is maintained on GitHub. Feel free to clone the repository.

```
git clone git://github.com/ping/instagram_private_api_extensions.git
```

1.2 Usage

1.2.1 media

```
from instagram_private_api import Client, MediaRatios
from instagram_private_api_extensions import media

api = Client('username', 'password')

# post a photo
photo_data, photo_size = media.prepare_image(
    'pathto/my_photo.jpg', aspect_ratios=MediaRatios.standard)
api.post_photo(photo_data, photo_size, caption='Hello World!')

# post a video
vid_data, vid_size, vid_duration, vid_thumbnail = media.prepare_video(
    'pathto/my_video.mp4', aspect_ratios=MediaRatios.standard)
api.post_video(vid_data, vid_size, vid_duration, vid_thumbnail)

# post a photo story
photo_data, photo_size = media.prepare_image(
    'pathto/my_photo.jpg', aspect_ratios=MediaRatios.reel)
api.post_photo_story(photo_data, photo_size)

# post a video story
vid_data, vid_size, vid_duration, vid_thumbnail = media.prepare_video(
    'pathto/my_video.mp4', aspect_ratios=MediaRatios.reel)
api.post_video_story(vid_data, vid_size, vid_duration, vid_thumbnail)

# post a video without reading the whole file into memory
vid_saved_path, vid_size, vid_duration, vid_thumbnail = media.prepare_video(
    'pathto/my_video.mp4', aspect_ratios=MediaRatios.standard,
    save_path='pathto/my_saved_video.mp4', save_only=True)
# To use save_only, the file must be saved locally
# by specifying the save_path
with open(vid_saved_path, 'rb') as video_fp:
    api.post_video(video_fp, vid_size, vid_duration, vid_thumbnail)
```

1.2.2 pagination

```
from instagram_private_api_extensions import pagination

# page through a feed
items = []
for results in pagination.page(api.user_feed, args={'user_id': '123456'}):
    if results.get('items'):
```

(continues on next page)

(continued from previous page)

```
        items.extend(results['items'])
print(len(items))
```

1.2.3 live

```
from instagram_private_api_extensions import live

broadcast = api.broadcast_info('1234567890')

dl = live.Downloader(
    mpd=broadcast['dash_playback_url'],
    output_dir='output_%s/' % str(broadcast['id']),
    user_agent=api.user_agent)
try:
    dl.run()
except KeyboardInterrupt:
    if not dl.is_aborted:
        dl.stop()
finally:
    # combine the downloaded files
    # Requires ffmpeg installed. If you prefer to use avconv
    # for example, omit this step and do it manually
    dl.stitch('my_video.mp4')
```

1.2.4 replay

```
from instagram_private_api_extensions import replay

user_story_feed = api.user_story_feed('12345')

broadcasts = user_story_feed.get('post_live_item', {}).get('broadcasts', [])
for broadcast in broadcasts:
    dl = replay.Downloader(
        mpd=broadcast['dash_manifest'],
        output_dir='output_{}/'.format(broadcast['id']),
        user_agent=api.user_agent)
    # download and save to file
    dl.download('output_{}.mp4'.format(broadcast['id']))
```

1.3 Developer Interface

This page of the documentation will cover all methods and classes available to the developer.

1.3.1 Media

instagram_private_api_extensions.media.**calc_crop**(*aspect_ratios*, *curr_size*)
Calculate if cropping is required based on the desired aspect ratio and the current size.

Parameters

- **aspect_ratios** – single float value or tuple of (min_ratio, max_ratio)
- **curr_size** – tuple of (width, height)

Returns

instagram_private_api_extensions.media.**calc_resize**(*max_size*, *curr_size*, *min_size=(0, 0)*)
Calculate if resize is required based on the max size desired and the current size

Parameters

- **max_size** – tuple of (width, height)
- **curr_size** – tuple of (width, height)
- **min_size** – tuple of (width, height)

Returns

instagram_private_api_extensions.media.**is_remote**(*media*)
Detect if media specified is a url

instagram_private_api_extensions.media.**prepare_image**(*img*, *max_size=(1080, 1350)*, *aspect_ratios=(0.8, 1.9148936170212767)*, *save_path=None*, ***kwargs*)
Prepares an image file for posting. Defaults for size and aspect ratio from https://help.instagram.com/1469029763400082

Parameters

- **img** – file path
- **max_size** – tuple of (max_width, max_height)
- **aspect_ratios** – single float value or tuple of (min_ratio, max_ratio)
- **save_path** – optional output file path
- **kwargs** –
  - **min_size**: tuple of (min_width, min_height)

Returns

instagram_private_api_extensions.media.**prepare_video**(*vid*, *thumbnail_frame_ts=0.0*, *max_size=(1080, 1350)*, *aspect_ratios=(0.8, 1.9148936170212767)*, *max_duration=60.0*, *save_path=None*, *skip_reencoding=False*, ***kwargs*)
Prepares a video file for posting. Defaults for size and aspect ratio from https://help.instagram.com/1469029763400082

Parameters

- **vid** – file path
- **thumbnail_frame_ts** – the frame of clip corresponding to time t (in seconds) to be used as the thumbnail
- **max_size** – tuple of (max_width, max_height)
- **aspect_ratios** – single float value or tuple of (min_ratio, max_ratio)
- **max_duration** – maximum video duration in seconds
- **save_path** – optional output video file path
- **skip_reencoding** – if set to True, the file will not be re-encoded if there are no modifications required. Default: False.
- **kwargs** –
  - **min_size**: tuple of (min_width, min_height)
  - **progress_bar**: bool flag to show/hide progress bar
  - **save_only**: bool flag to return only the path to the saved video file. Requires save_path be set.
  - **preset**: Sets the time that FFMPEG will spend optimizing the compression.

  Choices are: ultrafast, superfast, veryfast, faster, fast, medium, slow, slower, veryslow, placebo. Note that this does not impact the quality of the video, only the size of the video file. So choose ultrafast when you are in a hurry and file size does not matter.

Returns

1.3.2 Pagination

instagram_private_api_extensions.pagination.**page**(*fn*, *args*, *cursor_key='max_id'*, *get_cursor=<function <lambda>>*, *wait=5*)

A helper method to page through a feed/listing api call

```
from instagram_private_api import Client
from instagram_web_api import WebClient
from instagram_private_api_extensions.pagination import page

api = Client('username', 'password')
items = []
for results in page(api.user_feed, args={'user_id': '2958144170'}):
    if results.get('items'):
        items.extend(results['items'])
print(len(items))

webapi = WebClient(username='username', password='password', authenticate=True)
items = []
for results in pagination.page(
        webapi.user_feed,
        args={'user_id': '2958144170', 'extract': False},
        cursor_key='end_cursor',
        get_cursor=lambda r: r.get('media', {}).get('page_info', {}).get('end_
↪cursor')):

    if results.get('media', {}).get('nodes', []):
```

(continues on next page)

(continued from previous page)

```
        items.extend(results.get('media', {}).get('nodes', []))
print(len(items))
```

Parameters

- **fn** – function call
- **args** – dict of arguments to pass to fn
- **cursor_key** – param name for the cursor, e.g. 'max_id'
- **get_cursor** – anonymous function to etract the next cursor value
- **wait** – interval in seconds to sleep between api calls

Returns

1.3.3 Live

class instagram_private_api_extensions.live.**Downloader**(*mpd*, *output_dir*, *callback_check=None*, *singlethreaded=False*, *user_agent=None*, ***kwargs*)

Downloads and assembles a given IG live stream

__init__(*mpd*, *output_dir*, *callback_check=None*, *singlethreaded=False*, *user_agent=None*, ***kwargs*)

Parameters

- **mpd** – URL to mpd
- **output_dir** – folder to store the downloaded files
- **callback_check** – callback function that can be used to check on stream status if the downloader cannot be sure that the stream is over
- **singlethreaded** – flag to force single threaded downloads. Not advisable since this increases the probability of lost segments.

Returns

run()

Begin downloading

stitch(*output_filename*, *skipffmpeg=False*, *cleartempfiles=True*)

Combines all the dowloaded stream segments into the final mp4 file.

Parameters

- **output_filename** – Output file path
- **skipffmpeg** – bool flag to not use ffmpeg to join audio and video file into final mp4
- **cleartempfiles** – bool flag to remove downloaded and temp files

stop()

This is usually called automatically by the downloader but if the download process is interrupted unexpectedly, e.g. KeyboardInterrupt, you should call this method to gracefully close off the download.

Returns

1.3.4 Replay

class instagram_private_api_extensions.replay.**Downloader**(*mpd*, *output_dir*, *user_agent=None*, ***kwargs*)

Downloads and assembles a given IG live replay stream

__init__(*mpd*, *output_dir*, *user_agent=None*, ***kwargs*)

Parameters

- **mpd** – URL to mpd
- **output_dir** – folder to store the downloaded files

Returns

download(*output_filename*, *skipffmpeg=False*, *cleartempfiles=True*)

Download and saves the generated file with the file name specified.

Parameters

- **output_filename** – Output file path
- **skipffmpeg** – bool flag to not use ffmpeg to join audio and video file into final mp4
- **cleartempfiles** – bool flag to remove downloaded and temp files

Returns

Python Module Index

i

Index

Symbols

C

D

I

P

R

S