PyImgur Documentation

Release 0.5.3

Andreas Damgaard Pedersen

November 11, 2015

Contents

The simple way of using Imgur.

You can upload images, download images, read comments, update your albums, message people and more. In fact, you can do almost everything via PyImgur that you can via the webend.

CHAPTER 1

Other Content:

1.1 Authorization

Before we can do stuff on behalf of a user, such as uploading images to one of his albums. We need to authenticate as that user using a three step approach.

1.1.1 Step One - Ask for authorization

We cannot authenticate as a user, without first getting permission. So first we need to send the user to an Imgur url, where they can authorize our application to act on their behalf:

```
import webbrowser

import pyimgur

CLIENT_ID = "Your applications client_id"
CLIENT_SECRET = "Your applications client_secret"   # Needed for step 2 and 3

im = pyimgur.Imgur(CLIENT_ID, CLIENT_SECRET)
auth_url = im.authorization_url('pin')
webbrowser.open(auth_url)
pin = input("What is the pin? ") # Python 3x
#pin = raw_input("What is the pin? ") # Python 2x
```

The response argument in *authorization_url()* can be either 'pin' or 'code'. If it's code then the user will be redirected to your redirect url with the code as a get parameter after authorizing your application. If it's pin then after authorizing your application, the user will instead be shown a pin on Imgurs website.

1.1.2 Step Two - Exchange pin/code for access_token and refresh_token

Both pin and code are one-use token that we send to Imgur in exchange for a pair of access_token and refresh_token. These are used to authenticate as an user. Since we got a pin in the previous example, we'll use *exchange_pin()* to exchange it for an access_token and request_token. If we had gotten a code in the previous step, then we would have used *exchange_code()* instead:

```
im.exchange_pin(pin)
```

This returns an access_token and request_token and also updates `im`'s access_token and request_token attributes to the latest values. All requests to Imgur will now use this new authentication.

Now we can use our new authorization to do cool stuff, so let's create a new album for this user:

```
im.create_album("An authorized album", "Cool stuff!")
```

This line is exactly the same as how to create anonymous resources. But since we are now authenticated, it will create an album belonging to that user.

1.1.3 Step Three - Refreshing the access_token

access_tokens only last for 60 minutes, so every hour we'll need to update the access_token using the refresh_token. The refresh_token lasts forever and refreshing the access_token is easy:

```
im.refresh_access_token()
```

1.2 Changelog

This is the changelog for the released versions of PyImgur. These changes are divided into four categories.

- **[FEATURE]** Something new has been added.
- **[BUGFIX]** Something was broken before, but is now fixed.
- **[IMGUR]** A change caused by an upstream change from Imgur.
- **[CHANGE]** Other changes affecting user programs, such as the renaming of a function.

1.2.1 PyImgur 0.5.3

- **[FEATURE]** Make it possible to skip SSL certificate verification.
- **[FEATURE]** Increase python 2.6 compatibility. PyImgur is still only officially compatible with python 2.7.
- **[BUGFIX]** Fix missing *has_fetched* arguments breaking *get_subreddit_gallery()*.
- **[BUGFIX]** Fix bug breaking *get_gallery()* from working.
- **[BUGFIX]** Fix incorrect argument that prevented *exchange_code()* from working.

1.2.2 PyImgur 0.5.2

- **[BUGFIX]** Fixed an installation crash that happened if the *requests* dependency wasn't already installed.

1.2.3 PyImgur 0.5.1

- **[BUGFIX]** Fix bug in *update()* that caused it to crash when calling it with a list of image ids as the images argument. A bug also prevented the cover argument from being a Image object as is possible elsewhere, instead it could only be the id of an image.
- **[BUGFIX]** If an album had no cover Image, then before it would create a lazy Image object for the cover with `None` as Id. Now the `cover` attribute will correctly be `None`.
- **[BUGFIX]** Only albums instantiated with *get_album()* starts with the `images` attribute set. Now `_has_fetched` has been set to `False` for such albums. Meaning that a call to `Album.images` will refresh the object and it will then have the `images` attribute set.

1.2.4 PyImgur 0.5

- **[FEATURE]** Add *get_at_url()* that takes an url and returns an object matching what is located at the url.
- **[FEATURE]** Add *get_memes_gallery()* that return the gallery of memes as on the webend.
- **[FEATURE]** Add *get_subreddit_image()* that can return a subreddit image.
- **[IMGUR]** Imgur changed their API to return more data in the response, when uploading an image. But the variables that could be sent were always `None` in the response. See the bug report to Imgur
- **[BUGFIX]** If *download()* was used with an invalid filename given as the `name` argument or an invalid filename was gotten via the title, then the download would fail with an IOError. It now defaults to saving it with the hash as the name if the primary choice is an invalid filename.
- **[BUGFIX]** Manually calling *refresh()* didn't update the value of `_has_fetched`.

1.2.5 PyImgur 0.4.2

- **[FEATURE]** *upload_image()* can now upload images given with a url as well as being able to upload images given with a path. Either a path or a url to an image must be given when calling *upload_image()*.

1.2.6 PyImgur 0.4.1

- **[FEATURE]** Instead of returning an error, PyImgur will now resend requests to Imgur if it's expected that the second request will be successful. This is for the cases where Imgur has an internal error or the returned data is malformed.
- **[BUGFIX]** Fixed that User.get_images() unnecessarily required authentication as a user.

1.2.7 PyImgur 0.4.0

- **[CHANGE]** This version was a complete overhaul of PyImgur. It updated the version of Imgurs API PyImgur used to version 3.0 and implemented almost all functionality exposed. Additionally PyImgur changed from functional code to object oriented code.

1.3 Code Overview

1.3.1 `pyimgur` Package

PyImgur - The Simple Way of Using Imgur

PyImgur is a python wrapper of the popular image hosting and sharing website imgur.com. It makes the process of writing applications that uses Imgur faster, easier and less frustrating by automatically handling a lot of stuff for you. For instance you'll only need to use your client_id when you instantiate the Imgur object and when changing authentication. For the REST API this value needs to be sent with every request, but PyImgur handles this automatically.

Before using PyImgur, or the Imgur REST API in general, you'll need to register your application here: https://api.imgur.com/oauth2/addclient

For more information on usage visit https://github.com/Damgaard/PyImgur

class pyimgur.__init__.**Album**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

An album is a collection of images.

Variables

- **author** – The user that authored the album. None if anonymous.
- **cover** – The albums cover image.
- **datetime** – Time inserted into the gallery, epoch time.
- **deletehash** – For anonymous uploads, this is used to delete the album.
- **description** – A short description of the album.
- **id** – The ID for the album.
- **images** – A list of the images in this album. Only set at instantiation if created with Imgur.get_album. But even if it isn't set, then you can still access the attribute. This will make PyImgur fetch the newest version of all attributes for this class, including images. So it will work as though images was set all along.
- **is_favorited** – Has the logged in user favorited this album?
- **is_nsfw** – Is the album Not Safe For Work (contains gore/porn)?
- **layout** – The view layout of the album.
- **link** – The URL link to the album.
- **public** – The privacy level of the album, you can only view public albums if not logged in as the album owner.
- **section** – ??? - No info in Imgur documentation.
- **title** – The album's title
- **views** – Total number of views the album has received.

add_images(*images*)
Add images to the album.

Parameters **images** – A list of the images we want to add to the album. Can be Image objects, ids or a combination of the two. Images that you cannot add (non-existing or not owned by you) will not cause exceptions, but fail silently.

delete()
Delete this album.

favorite()
Favorite the album.

Favoriting an already favorited album will unfavor it.

remove_images(*images*)
Remove images from the album.

Parameters **images** – A list of the images we want to remove from the album. Can be Image objects, ids or a combination of the two. Images that you cannot remove (non-existing, not owned by you or not part of album) will not cause exceptions, but fail silently.

set_images(*images*)
Set the images in this album.

Parameters **images** – A list of the images we want the album to contain. Can be Image objects, ids or a combination of the two. Images that images that you cannot set (non-existing or not owned by you) will not cause exceptions, but fail silently.

submit_to_gallery(*title*, *bypass_terms=False*)
Add this to the gallery.

Require that the authenticated user has accepted gallery terms and verified their email.

Parameters

- **title** – The title of the new gallery item.
- **bypass_terms** – If the user has not accepted Imgur's terms yet, this method will return an error. Set this to True to by-pass the terms.

update(*title=None*, *description=None*, *images=None*, *cover=None*, *layout=None*, *privacy=None*)
Update the album's information.

Arguments with the value None will retain their old values.

Parameters

- **title** – The title of the album.
- **description** – A description of the album.
- **images** – A list of the images we want the album to contain. Can be Image objects, ids or a combination of the two. Images that images that you cannot set (non-existing or not owned by you) will not cause exceptions, but fail silently.
- **privacy** – The albums privacy level, can be public, hidden or secret.
- **cover** – The id of the cover image.
- **layout** – The way the album is displayed, can be blog, grid, horizontal or vertical.

class pyimgur.__init__.**Basic_object**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: object

Contains basic functionality shared by a lot of PyImgur's classes.

refresh()
Refresh this objects attributes to the newest values.

Attributes that weren't added to the object before, due to lazy loading, will be added by calling refresh.

class pyimgur.__init__.**Comment**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

A comment a user has made.

Users can comment on Gallery album, Gallery image or other Comments.

Variables

- **album_cover** – If this Comment is on a Album, this will be the Albums cover Image.
- **author** – The user that created the comment.
- **datetime** – Time inserted into the gallery, epoch time.
- **deletehash** – For anonymous uploads, this is used to delete the image.
- **downs** – The total number of dislikes (downvotes) the comment has received.
- **image** – The image the comment belongs to.

- **is_deleted** – Has the comment been deleted?
- **on_album** – Is the image part of an album.
- **parent** – The comment this one has replied to, if it is a top-level comment i.e. it's a comment directly to the album / image then it will be None.
- **permalink** – A permanent link to the comment.
- **points** – ups - downs.
- **replies** – A list of comment replies to this comment. This variable is only available if the comment was returned via Album.get_comments(). Use get_replies instead to get the replies if this variable is not available.
- **text** – The comments text.
- **ups** – The total number of likes (upvotes) the comment has received.
- **vote** – The currently logged in users vote on the comment.

delete()
Delete the comment.

downvote()
Downvote this comment.

get_replies()
Get the replies to this comment.

reply(*text*)
Make a comment reply.

upvote()
Upvote this comment.

class pyimgur.__init__.**Gallery_album**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Album*, *pyimgur.__init__.Gallery_item*

Gallery Albums are albums submitted to the gallery.

class pyimgur.__init__.**Gallery_image**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Image*, *pyimgur.__init__.Gallery_item*

Gallery images are images submitted to the gallery.

class pyimgur.__init__.**Gallery_item**
Bases: object

Functionality shared by Gallery_image and Gallery_album.

comment(*text*)
Make a top-level comment to this.

Parameters **text** – The comment text.

downvote()
Dislike this.

A downvote will replace a neutral vote or an upvote. Downvoting something the authenticated user has already downvoted will set the vote to neutral.

get_comments()
Get a list of the top-level comments.

remove_from_gallery()
Remove this image from the gallery.

upvote()
Like this.

An upvote will replace a neutral vote or an downvote. Upvoting something the authenticated user has already upvoted will set the vote to neutral.

class pyimgur.__init__.**Image**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

An image uploaded to Imgur.

Variables

- **bandwidth** – Bandwidth consumed by the image in bytes.
- **datetime** – Time inserted into the gallery, epoch time.
- **deletehash** – For anonymous uploads, this is used to delete the image.
- **description** – A short description of the image.
- **height** – The height of the image in pixels.
- **id** – The ID for the image.
- **is_animated** – is the image animated?
- **is_favorited** – Has the logged in user favorited this album?
- **is_nsfw** – Is the image Not Safe For Work (contains gore/porn)?
- **link** – The URL link to the image.
- **link_big_square** – The URL to a big square thumbnail of the image.
- **link_huge_thumbnail** – The URL to a huge thumbnail of the image.
- **link_large_square** – The URL to a large square thumbnail of the image.
- **link_large_thumbnail** – The URL to a large thumbnail of the image.
- **link_medium_thumbnail** – The URL to a medium thumbnail of the image.
- **link_small_square** – The URL to a small square thumbnail of the image.
- **section** – ??? - No info in Imgur documentation.
- **size** – The size of the image in bytes.
- **title** – The albums title.
- **views** – Total number of views the album has received.
- **width** – The width of the image in bytes.

delete()
Delete the image.

download(*path=''*, *name=None*, *overwrite=False*, *size=None*)
Download the image.

Parameters

- **path** – The image will be downloaded to the folder specified at path, if path is None (default) then the current working directory will be used.

- **name** – The name the image will be stored as (not including file extension). If name is None, then the title of the image will be used. If the image doesn't have a title, it's id will be used. Note that if the name given by name or title is an invalid filename, then the hash will be used as the name instead.
- **overwrite** – If True overwrite already existing file with the same name as what we want to save the file as.
- **size** – Instead of downloading the image in it's original size, we can choose to instead download a thumbnail of it. Options are 'small_square', 'big_square', 'small_thumbnail', 'medium_thumbnail', 'large_thumbnail' or 'huge_thumbnail'.

Returns Name of the new file.

favorite()
Favorite the image.

Favoriting an already favorited image will unfavorite it.

submit_to_gallery(*title*, *bypass_terms=False*)
Add this to the gallery.

Require that the authenticated user has accepted gallery terms and verified their email.

Parameters

- **title** – The title of the new gallery item.
- **bypass_terms** – If the user has not accepted Imgur's terms yet, this method will return an error. Set this to True to by-pass the terms.

update(*title=None*, *description=None*)
Update the image with a new title and/or description.

class pyimgur.__init__.**Imgur**(*client_id*, *client_secret=None*, *access_token=None*, *refresh_token=None*, *verify=True*)
The base class containing general functionality for Imgur.

You should create an Imgur object at the start of your code and use it to interact with Imgur. You shouldn't directly initialize any other classes, but instead use the methods in this class to get them.

Initialize the Imgur object.

Before using PyImgur, or the Imgur REST API in general, you need to register your application with Imgur. This can be done at https://api.imgur.com/oauth2/addclient

Parameters

- **client_id** – Your applications client_id.
- **client_secret** – Your applications client_secret. This is only needed when a user needs to authorize the app.
- **access_token** – is your secret key used to access the user's data. It can be thought of the user's password and username combined into one, and is used to access the user's account. It expires after 1 hour.
- **refresh_token** – is used to request new access_tokens. Since access_tokens expire after 1 hour, we need a way to request new ones without going through the entire authorization step again. It does not expire.
- **verify** – Verify SSL certificate of server (can result in SSLErrors)?

authorization_url(*response*, *state=''*)
Return the authorization url that's needed to authorize as a user.

Parameters

- **response** – Can be either code or pin. If it's code the user will be redirected to your redirect url with the code as a get parameter after authorizing your application. If it's pin then after authorizing your application, the user will instead be shown a pin on Imgurs website. Both code and pin are used to get an access_token and refresh token with the exchange_code and exchange_pin functions respectively.
- **state** – This optional parameter indicates any state which may be useful to your application upon receipt of the response. Imgur round-trips this parameter, so your application receives the same value it sent. Possible uses include redirecting the user to the correct resource in your site, nonces, and cross-site-request-forgery mitigations.

change_authentication(*client_id=None*, *client_secret=None*, *access_token=None*, *refresh_token=None*)
Change the current authentication.

create_album(*title=None*, *description=None*, *images=None*, *cover=None*)
Create a new Album.

Parameters

- **title** – The title of the album.
- **description** – The albums description.
- **images** – A list of the images that will be added to the album after it's created. Can be Image objects, ids or a combination of the two. Images that you cannot add (non-existing or not owned by you) will not cause exceptions, but fail silently.
- **cover** – The id of the image you want as the albums cover image.

Returns The newly created album.

exchange_code(*code*)
Exchange one-use code for an access_token and request_token.

exchange_pin(*pin*)
Exchange one-use pin for an access_token and request_token.

get_album(*id*)
Return information about this album.

get_at_url(*url*)
Return a object representing the content at url.

Returns None if no object could be matched with the id.

Works for Album, Comment, Gallery_album, Gallery_image, Image and User.

NOTE: Imgur's documentation does not cover what urls are available. Some urls, such as imgur.com/<ID> can be for several different types of object. Using a wrong, but similair call, such as get_subreddit_image on a meme image will not cause an error. But instead return a subset of information, with either the remaining pieces missing or the value set to None. This makes it hard to create a method such as this that attempts to deduce the object from the url. Due to these factors, this method should be considered experimental and used carefully.

Parameters **url** – The url where the content is located at

get_comment(*id*)
Return information about this comment.

get_gallery(*section='hot'*, *sort='viral'*, *window='day'*, *show_viral=True*, *limit=None*)
Return a list of gallery albums and gallery images.

Parameters

- **section** – hot | top | user - defaults to hot.
- **sort** – viral | time - defaults to viral.
- **window** – Change the date range of the request if the section is "top", day | week | month | year | all, defaults to day.
- **show_viral** – true | false - Show or hide viral images from the 'user' section. Defaults to true.
- **limit** – The number of items to return.

get_gallery_album(*id*)
Return the gallery album matching the id.

Note that an album's id is different from it's id as a gallery album. This makes it possible to remove an album from the gallery and setting it's privacy setting as secret, without compromising it's secrecy.

get_gallery_image(*id*)
Return the gallery image matching the id.

Note that an image's id is different from it's id as a gallery image. This makes it possible to remove an image from the gallery and setting it's privacy setting as secret, without compromising it's secrecy.

get_image(*id*)
Return a Image object representing the image with the given id.

get_memes_gallery(*sort='viral'*, *window='week'*, *limit=None*)
Return a list of gallery albums/images submitted to the memes gallery

The url for the memes gallery is: http://imgur.com/g/memes

Parameters

- **sort** – viral | time | top - defaults to viral
- **window** – Change the date range of the request if the section is "top", day | week | month | year | all, defaults to week.
- **limit** – The number of items to return.

get_message(*id*)
Return a Message object for given id.

Parameters **id** – The id of the message object to return.

get_notification(*id*)
Return a Notification object.

Parameters **id** – The id of the notification object to return.

get_subreddit_gallery(*subreddit*, *sort='time'*, *window='top'*, *limit=None*)
Return a list of gallery albums/images submitted to a subreddit.

A subreddit is a subsection of the website www.reddit.com, where users can, among other things, post images.

Parameters

- **subreddit** – A valid subreddit name.
- **sort** – time | top - defaults to top.
- **window** – Change the date range of the request if the section is "top", day | week | month | year | all, defaults to day.

- **limit** – The number of items to return.

get_subreddit_image(*subreddit*, *id*)
Return the Gallery_image with the id submitted to subreddit gallery

Parameters

- **subreddit** – The subreddit the image has been submitted to.
- **id** – The id of the image we want.

get_user(*username*)
Return a User object for this username.

Parameters **username** – The name of the user we want more information about.

is_imgur_url(*url*)
Is the given url a valid Imgur url?

refresh_access_token()
Refresh the access_token.

The self.access_token attribute will be updated with the value of the new access_token which will also be returned.

search_gallery(*q*)
Search the gallery with the given query string.

upload_image(*path=None*, *url=None*, *title=None*, *description=None*, *album=None*)
Upload the image at either path or url.

Parameters

- **path** – The path to the image you want to upload.
- **url** – The url to the image you want to upload.
- **title** – The title the image will have when uploaded.
- **description** – The description the image will have when uploaded.
- **album** – The album the image will be added to when uploaded. Can be either a Album object or it's id. Leave at None to upload without adding to an Album, adding it later is possible. Authentication as album owner is necessary to upload to an album with this function.

Returns An Image object representing the uploaded image.

class pyimgur.__init__.**Message**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

This corresponds to the messages users can send each other.

delete()
Delete the message.

get_thread()
Return the message thread this Message is in.

reply(*body*)
Reply to this message.

This is a convenience method calling User.send_message. See it for more information on usage. Note that both recipient and reply_to are given by using this convenience method.

Parameters **body** – The body of the message.

class pyimgur.__init__.**Notification**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

This corresponds to the notifications a user may receive.

A notification can come for several reasons. For instance, one may be received if someone replies to one of your comments.

mark_as_viewed()
Mark the notification as viewed.

Notifications cannot be marked as unviewed.

class pyimgur.__init__.**User**(*json_dict*, *imgur*, *has_fetched=True*)
Bases: *pyimgur.__init__.Basic_object*

A User on Imgur.

Variables

- **bio** – A basic description filled out by the user, displayed in the gallery profile page.
- **created** – The epoch time of user account creation
- **id** – The user id.
- **name** – The username
- **reputation** – Total likes - dislikes of the user's created content.

change_settings(*bio=None*, *public_images=None*, *messaging_enabled=None*, *album_privacy=None*, *accepted_gallery_terms=None*)
Update the settings for the user.

Parameters

- **bio** – A basic description filled out by the user, is displayed in the gallery profile page.
- **public_images** – Set the default privacy setting of the users images. If True images are public, if False private.
- **messaging_enabled** – Set to True to enable messaging.
- **album_privacy** – The default privacy level of albums created by the user. Can be public, hidden or secret.
- **accepted_gallery_terms** – The user agreement to Imgur Gallery terms. Necessary before the user can submit to the gallery.

delete()
Delete this user. Require being authenticated as the user.

get_albums(*limit=None*)
Return a list of the user's albums.

Secret and hidden albums are only returned if this is the logged-in user.

get_comments()
Return the comments made by the user.

get_favorites()
Return the users favorited images.

get_gallery_favorites()
Get a list of the images in the gallery this user has favorited.

get_gallery_profile()
Return the users gallery profile.

get_images(*limit=None*)
Return all of the images associated with the user.

get_messages(*new=True*)
Return all messages sent to this user, formatted as a notification.

Parameters **new** – False for all notifications, True for only non-viewed notifications.

get_notifications(*new=True*)
Return all the notifications for this user.

get_replies(*new=True*)
Return all reply notifications for this user.

Parameters **new** – False for all notifications, True for only non-viewed notifications.

get_settings()
Returns current settings.

Only accessible if authenticated as the user.

get_statistics()
Return statistics about this user.

get_submissions(*limit=None*)
Return a list of the images a user has submitted to the gallery.

has_verified_email()
Has the user verified that the email he has given is legit?

Verified e-mail is required to the gallery. Confirmation happens by sending an email to the user and the owner of the email user verifying that he is the same as the Imgur user.

send_message(*body*, *subject=None*, *reply_to=None*)
Send a message to this user from the logged in user.

Parameters

- **body** – The body of the message.
- **subject** – The subject of the message. Note that if the this message is a reply, then the subject of the first message will be used instead.
- **reply_to** – Messages can either be replies to other messages or start a new message thread. If this is None it will start a new message thread. If it's a Message object or message_id, then the new message will be sent as a reply to the reply_to message.

send_verification_email()
Send verification email to this users email address.

Remember that the verification email may end up in the users spam folder.

CHAPTER 2

Installation

The recommended way to install is via pip

```
$ pip install pyimgur
```

CHAPTER 3

Getting Started

Before we can start using PyImgur, we need to register our application with Imgur. This way, Imgur can see what each application is doing on their site. Go to https://api.imgur.com/oauth2/addclient to register your client. Note that you can't use an application registration for the old v2 version of the Imgur API, which was depreciated December 2012.

When we registered our application we got a `client_id` and a `client_secret`. The `client_secret` is used for authenticating as a user, if we just need access to public or anonymous resources, then we can leave it out. For our first example we're going to get some information about an image already uploaded to image:

```
import pyimgur
CLIENT_ID = "Your_applications_client_id"
im = pyimgur.Imgur(CLIENT_ID)
image = im.get_image('S1jmapR')
print(image.title) # Cat Ying & Yang
print(image.link) # http://imgur.com/S1jmapR.jpg
```

The `Imgur` object keeps the authentication information, changes authentication and is the common way to get objects from Imgur.

CHAPTER 4

Uploading an Image

Let's use another example to show how to upload an image:

```
import pyimgur

CLIENT_ID = "Your_applications_client_id"
PATH = "A Filepath to an image on your computer"

im = pyimgur.Imgur(CLIENT_ID)
uploaded_image = im.upload_image(PATH, title="Uploaded with PyImgur")
print(uploaded_image.title)
print(uploaded_image.link)
print(uploaded_image.size)
print(uploaded_image.type)
```

Some methods here one or more arguments with the default value `None`. For methods modifying existing objects, this mean to keep the already existing value. For methods not modifying existing objects, this mean to use the Imgur default.

CHAPTER 5

Lazy objects

To reduce the load on Imgur, PyImgur only requests the data it needs. This means each object has the attribute `_has_fetched` which if `True` ` has fetched all the data it can, if `False` it can fetch more information.

Whenever we request an attribute that hasn't been loaded the newest information will be requested from Imgur and all the object attributes will be updated to the newest values. We can also use the method `refresh()` to force a call to Imgur, that will update the object with the latest values:

```
import pyimgur
CLIENT_ID = "Your_applications_client_id"
im = pyimgur.Imgur(CLIENT_ID)
gallery_image = im.get_gallery_image('JiAaT')
author = gallery_image.author
print(author._has_fetched) # False ie. it's a lazily loaded object
print(author.reputation)
print(author._has_fetched) # True ie. all values have now been retrieved.
```

CHAPTER 6

Introspection

Remember that as usual you can use the `dir`, `vars` and `help` builtin functions to introspect objects to learn more about them and how they work.

CHAPTER 7

Support

If you find an bug, have any questions about how to use PyImgur or have suggestions for improvements then feel free to file an issue on the Github project page.

CHAPTER 8

Documentation

PyImgur's full documentation is located on ReadTheDocs.

CHAPTER 9

License

All of the code contained here is licensed by the GNU GPLv3.

p