Mastodon.py Documentation

Release 1.2.1

Lorenz Diener

Dec 21, 2017

Contents

```
from mastodon import Mastodon

# Register app - only once!
'''
Mastodon.create_app(
     'pytooterapp',
     api_base_url = 'https://mastodon.social',
     to_file = 'pytooter_clientcred.secret'
)
'''

# Log in - either every time, or use persisted
'''
mastodon = Mastodon(
    client_id = 'pytooter_clientcred.secret',
    api_base_url = 'https://mastodon.social'
)
mastodon.log_in(
    'my_login_email@example.com',
    'incrediblygoodpassword',
    to_file = 'pytooter_usercred.secret'
)
'''

# Create actual API instance
mastodon = Mastodon(
    client_id = 'pytooter_clientcred.secret',
    access_token = 'pytooter_usercred.secret',
    api_base_url = 'https://mastodon.social'
)
mastodon.toot('Tooting from python using #mastodonpy !')
```

Mastodon is an ActivityPub and OStatus based twitter-like federated social network node. It has an API that allows you to interact with its every aspect. This is a simple python wrapper for that api, provided as a single python module. By default, it talks to the Mastodon flagship instance, but it can be set to talk to any node running Mastodon by setting *api_base_url* when creating the api object (or creating an app).

Mastodon.py aims to implement the complete public Mastodon API. As of this time, it is feature complete for Mastodon version 2.1.0.

CHAPTER 1

A note about rate limits

Mastodons API rate limits per user account. By default, the limit is 300 requests per 5 minute time slot. This can differ from instance to instance and is subject to change. Mastodon.py has three modes for dealing with rate limiting that you can pass to the constructor, “throw”, “wait” and “pace”, “wait” being the default.

In “throw” mode, Mastodon.py makes no attempt to stick to rate limits. When a request hits the rate limit, it simply throws a *MastodonRateLimitError*. This is for applications that need to handle all rate limiting themselves (i.e. interactive apps), or applications wanting to use Mastodon.py in a multi-threaded context (“wait” and “pace” modes are not thread safe).

In “wait” mode, once a request hits the rate limit, Mastodon.py will wait until the rate limit resets and then try again, until the request succeeds or an error is encountered. This mode is for applications that would rather just not worry about rate limits much, don’t poll the api all that often, and are okay with a call sometimes just taking a while.

In “pace” mode, Mastodon.py will delay each new request after the first one such that, if requests were to continue at the same rate, only a certain fraction (set in the constructor as *ratelimit_pacefactor*) of the rate limit will be used up. The fraction can be (and by default, is) greater than one. If the rate limit is hit, “pace” behaves like “wait”. This mode is probably the most advanced one and allows you to just poll in a loop without ever sleeping at all yourself. It is for applications that would rather just pretend there is no such thing as a rate limit and are fine with sometimes not being very interactive.

In addition to the per-user limit, there is a per-IP limit of 7500 requests per 5 minute time slot, and tighter limits on logins. Mastodon.py does not make any effort to respect these.

If your application requires many hits to endpoints that are available without logging in, do consider using Mastodon.py without authenticating to get the full per-IP limit. In this case, you can set the Mastodon objects *ratelimit_limit* and *ratelimit_remaining* properties appropriately if you want to use advanced rate limit handling.

CHAPTER 2

A note about pagination

Many of Mastodons API endpoints are paginated. What this means is that if you request data from them, you might not get all the data at once - instead, you might only get the first few results.

All endpoints that are paginated have three parameters: since_id, max_id and limit. since_id allows you to specify the smallest id you want in the returned data. max_id, similarly, allows you to specify the largest. By specifying either one (generally, only one, not both) of them you can go through pages forwards and backwards.

limit allows you to specify how many results you would like returned. Note that an instance may choose to return less results than you requested.

The responses returned by paginated endpoints contain a "link" header that specifies which parameters to use to get the next and previous pages. Mastodon.py parses these and stores them (if present) in the first (for the previous page) and last (for the next page) item of the returned list as _pagination_prev and _pagination_next.

There are convenience functions available for fetching the previous and next page of a paginated request as well as for fetching all pages starting from a first page.

CHAPTER 3

Two notes about IDs

Mastodons API uses IDs in several places: User IDs, Toot IDs, ...

While debugging, it might be tempting to copy-paste in IDs from the web interface into your code. This will not work, as the IDs on the web interface and in the URLs are not the same as the IDs used internally in the API, so don't do that.

3.1 ID unpacking

Wherever Mastodon.py expects an ID as a parameter, you can also pass a dict that contains an id - this means that, for example, instead of writing

```
mastodon.status_post("@somebody wow!", in_reply_to_id = toot["id"])
```

you can also just write

```
mastodon.status_post("@somebody wow!", in_reply_to_id = toot)
```

and everything will work as intended.

CHAPTER 4

Error handling

When Mastodon.py encounters an error, it will raise an exception, generally with some text included to tell you what went wrong.

The base class that all mastodon exceptions inherit from is *MastodonError*. If you are only interested in the fact an error was raised somewhere in Mastodon.py, and not the details, this is the exception you can catch.

MastodonIllegalArgumentError is generally a programming problem - you asked the API to do something obviously invalid (i.e. specify a privacy option that does not exist).

MastodonFileNotFoundError and *MastodonNetworkError* are IO errors - could be you specified a wrong URL, could be the internet is down or your hard drive is dying. They inherit from MastodonIOError, for easy catching.

MastodonAPIError is an error returned from the Mastodon instance - the server has decided it can't fullfill your request (i.e. you requested info on a user that does not exist).

MastodonRatelimitError is raised when you hit an API rate limit. You should try again after a while (see the rate limiting section above).

MastodonVersionError is raised when a version check for an API call fails.

CHAPTER 5

Return values

Unless otherwise specified, all data is returned as python dictionaries, matching the JSON format used by the API. Dates returned by the API are in ISO 8601 format and are parsed into python datetime objects.

To make access easier, the dictionaries returned are wrapped by a class that adds read-only attributes for all dict values - this means that, for example, instead of writing

```
description = mastodon.account_verify_credentials()["source"]["note"]
```

you can also just write

```
description = mastodon.account_verify_credentials().source.note
```

and everything will work as intended.

5.1 User dicts

```
mastodon.account(<numerical id>)
# Returns the following dictionary:
{
    'id': # Same as <numerical id>
    'username': # The username (what you @ them with)
    'acct': # The user's account name as username@domain (@domain omitted for local␣
↪users)
    'display_name': # The user's display name
    'locked': # Denotes whether the account can be followed without a follow request
    'created_at': # Account creation time
    'following_count': # How many people they follow
    'followers_count': # How many followers they have
    'statuses_count': # How many statuses they have
    'note': # Their bio
    'url': # Their URL; usually 'https://mastodon.social/users/<acct>'
    'avatar': # URL for their avatar, can be animated
    'header': # URL for their header image, can be animated
```

```
    'avatar_static': # URL for their avatar, never animated
    'header_static': # URL for their header image, never animated
    'source': # Additional information - only present for user dict returned
              # from account_verify_credentials()
    'moved_to_account': # If set, an account dict of the account this user has
                        # set up as their moved-to address.
}

mastodon.account_verify_credentials()["source"]
# Returns the following dictionary:
{
    'privacy': # The users default visibility setting ("private", "unlisted" or
↪"public")
    'sensitive': # Denotes whether user media should be marked sensitive by default
    'note': # Plain text version of the users bio
}
```

5.2 Toot dicts

```
mastodon.toot("Hello from Python")
# Returns the following dictionary:
{
    'id': # Numerical id of this toot
    'uri': # Descriptor for the toot
        # EG 'tag:mastodon.social,2016-11-25:objectId=<id>:objectType=Status'
    'url': # URL of the toot
    'account': # User dict for the account which posted the status
    'in_reply_to_id': # Numerical id of the toot this toot is in response to
    'in_reply_to_account_id': # Numerical id of the account this toot is in response␣
↪to
    'reblog': # Denotes whether the toot is a reblog. If so, set to the original toot␣
↪dict.
    'content': # Content of the toot, as HTML: '<p>Hello from Python</p>'
    'created_at': # Creation time
    'reblogs_count': # Number of reblogs
    'favourites_count': # Number of favourites
    'reblogged': # Denotes whether the logged in user has boosted this toot
    'favourited': # Denotes whether the logged in user has favourited this toot
    'sensitive': # Denotes whether media attachments to the toot are marked sensitive
    'spoiler_text': # Warning text that should be displayed before the toot content
    'visibility': # Toot visibility ('public', 'unlisted', 'private', or 'direct')
    'mentions': # A list of users dicts mentioned in the toot, as Mention dicts
    'media_attachments': # A list of media dicts of attached files
    'emojis': # A list of custom emojis used in the toot, as Emoji dicts
    'tags': # A list of hashtag used in the toot, as Hashtag dicts
    'application': # Application dict for the client used to post the toot (Does not␣
↪federate
                   # and is therefore always None for remote toots, can also be None␣
↪for
                   # local toots for some legacy applications).
    'language': # The language of the toot, if specified by the server.
    'muted': # Boolean denoting whether the user has muted this status by
             # way of conversation muting
}
```

5.3 Mention dicts

```
{
    'url': # Mentioned users profile URL (potentially remote)
    'username': # Mentioned users user name (not including domain)
    'acct': # Mentioned users account name (including domain)
    'id': # Mentioned users (local) account ID
}
```

5.4 Hashtag dicts

```
{
    'name': # Hashtag name (not including the #)
    'url': # Hashtag URL (can be remote)
}
```

5.5 Emoji dicts

```
{
    'shortcode': # Emoji shortcode, without surrounding colons
    'url': # URL for the emoji image, can be animated
    'static_url': # URL for the emoji image, never animated
}
```

5.6 Application dicts

```
{
    'name': # The applications name
    'website': # The applications website
}
```

5.7 Relationship dicts

```
mastodon.account_follow(<numerical id>)
# Returns the following dictionary:
{
    'id': # Numerical id (same one as <numerical id>)
    'following': # Boolean denoting whether the logged-in user follows the specified␣
↪user
    'followed_by': # Boolean denoting whether the specified user follows the logged-
↪in user
    'blocking': # Boolean denoting whether the logged-in user has blocked the␣
↪specified user
    'muting': # Boolean denoting whether the logged-in user has muted the specified␣
↪user
```

```
    'requested': # Boolean denoting whether the logged-in user has sent the specified
                 # user a follow request
    'domain_blocking': # Boolean denoting whether the logged-in user has blocked the
                       # specified users domain
}
```

5.8 Notification dicts

```
mastodon.notifications()[0]
# Returns the following dictionary:
{
    'id': # id of the notification
    'type': # "mention", "reblog", "favourite" or "follow"
    'created_at': # The time the notification was created
    'account': # User dict of the user from whom the notification originates
    'status': # In case of "mention", the mentioning status
              # In case of reblog / favourite, the reblogged / favourited status
}
```

5.9 Context dicts

```
mastodon.status_context(<numerical id>)
# Returns the following dictionary:
{
    'ancestors': # A list of toot dicts
    'descendants': # A list of toot dicts
}
```

5.10 List dicts

```
mastodon.list(<numerical id>)
# Returns the following dictionary:
{
    'id': # id of the list
    'title': # title of the list
}
```

5.11 Media dicts

```
mastodon.media_post("image.jpg", "image/jpeg")
# Returns the following dictionary:
{
    'id': # The ID of the attachment.
    'type': # Media type: 'image', 'video', 'gifv' or 'unknown'.
    'url': # The URL for the image in the local cache
    'remote_url': # The remote URL for the media (if the image is from a remote␣
↪instance)
```

```
    'preview_url': # The URL for the media preview
    'text_url': # The display text for the media (what shows up in toots)
    'meta': # Dictionary of two image metadata dicts (see below),
            # 'original' and 'small' (preview)
}

# Metadata dicts:
{
   'width': # Width of the image in pixels
   'height': # Height of the image in pixels
   'aspect': # Aspect ratio of the image as a floating point number
   'size': # Textual representation of the image size in pixels, e.g. '800x600'
}
```

5.12 Card dicts

```
mastodon.status_card(<numerical id>):
# Returns the following dictionary
{
    'url': # The URL of the card.
    'title': # The title of the card.
    'description': # The description of the card.
    'type': # Embed type: 'link', 'photo', 'video', or 'rich'
    'image': # (optional) The image associated with the card.

    # OEmbed data (all optional):
    'author_name': # Name of the embedded contents author
    'author_url': # URL pointing to the embedded contents author
    'description': # Description of the embedded content
    'width': # Width of the embedded object
    'height': # Height of the embedded object
    'html': # HTML string of the embed
    'provider_name': # Name of the provider from which the embed originates
    'provider_url': # URL pointing to the embeds provider
}
```

5.13 Search result dicts

```
mastodon.search("<query>")
# Returns the following dictionary
{
    'accounts': # List of account dicts resulting from the query
    'hashtags': # List of hashtag dicts resulting from the query
    'statuses': # List of toot dicts resulting from the query
}
```

5.14 Instance dicts

```
mastodon.instance()
# Returns the following dictionary
{
    'description': # A brief instance description set by the admin
    'email': # The admin contact e-mail
    'title': # The instances title
    'uri': # The instances URL
    'version': # The instances mastodon version
    'urls': # Additional URLs dict, presently only 'streaming_api' with the
            # stream websocket address.
}
```

5.15 Report dicts

```
mastodon.reports()[0]
# Returns the following dictionary
{
    'id': # Numerical id of the report
    'action_taken': # True if a moderator or admin has processed the
                    # report, False otherwise. Note that no indication as to
                    # what action was taken is given and that an admin simply
                    # marking the report as processed and not doing anything else
                    # will set this field to True.
}
```

CHAPTER 6

App registration and user authentication

Before you can use the mastodon API, you have to register your application (which gets you a client key and client secret) and then log in (which gets you an access token). These functions allow you to do those things. For convenience, once you have a client id, secret and access token, you can simply pass them to the constructor of the class, too!

Note that while it is perfectly reasonable to log back in whenever your app starts, registering a new application on every startup is not, so don't do that - instead, register an application once, and then persist your client id and secret. A convenient method for this is provided by the functions dealing with registering the app, logging in and the Mastodon classes constructor.

To talk to an instance different from the flagship instance, specify the api_base_url (usually, just the URL of the instance, i.e. https://mastodon.social/ for the flagship instance). If no protocol is specified, Mastodon.py defaults to https.

static Mastodon.**create_app**(*client_name, scopes=['read', 'write', 'follow'], redirect_uris=None, website=None, to_file=None, api_base_url='https://mastodon.social', request_timeout=300*)

Create a new app with given *client_name* and *scopes* (read, write, follow)

Specify *redirect_uris* if you want users to be redirected to a certain page after authenticating. Specify *to_file* to persist your apps info to a file so you can use them in the constructor. Specify *api_base_url* if you want to register an app on an instance different from the flagship one.

Presently, app registration is open by default, but this is not guaranteed to be the case for all future mastodon instances or even the flagship instance in the future.

Returns *client_id* and *client_secret*, both as strings.

Mastodon.**__init__**(*client_id*, *client_secret=None*, *access_token=None*, *api_base_url='https://mastodon.social'*, *debug_requests=False*, *ratelimit_method='wait'*, *ratelimit_pacefactor=1.1*, *request_timeout=300*, *mastodon_version=None*, *version_check_mode='created'*)

Create a new API wrapper instance based on the given *client_secret* and *client_id*. If you give a *client_id* and it is not a file, you must also give a secret.

You can also specify an *access_token*, directly or as a file (as written by *log_in()*).

Mastodon.py can try to respect rate limits in several ways, controlled by *ratelimit_method*. "throw" makes functions throw a *MastodonRatelimitError* when the rate limit is hit. "wait" mode will, once the limit is hit, wait and retry the request as soon as the rate limit resets, until it succeeds. "pace" works like throw, but tries to wait in between calls so that the limit is generally not hit (How hard it tries to not hit the rate limit can be controlled by ratelimit_pacefactor). The default setting is "wait". Note that even in "wait" and "pace" mode, requests can still fail due to network or other problems! Also note that "pace" and "wait" are NOT thread safe.

Specify *api_base_url* if you wish to talk to an instance other than the flagship one. If a file is given as *client_id*, client ID and secret are read from that file.

By default, a timeout of 300 seconds is used for all requests. If you wish to change this, pass the desired timeout (in seconds) as *request_timeout*.

The *mastodon_version* parameter can be used to specify the version of Mastodon that Mastodon.py will expect to be installed on the server. The function will throw an error if an unparseable Version is specified. If no version is specified, Mastodon.py will set *mastodon_version* to the detected version.

The version check mode can be set to "created" (the default behaviour), "changed" or "none". If set to "created", Mastodon.py will throw an error if the version of Mastodon it is connected to is too old to have an endpoint. If it is set to "changed", it will throw an error if the endpoints behaviour has changed after the version of Mastodon that is connected has been released. If it is set to "none", version checking is disabled.

`Mastodon.`**`log_in`**(*username=None, password=None, code=None, redirect_uri='urn:ietf:wg:oauth:2.0:oob', refresh_token=None, scopes=['read', 'write', 'follow'], to_file=None*)

Get the access token for a user.

The username is the e-mail used to log in into mastodon.

Can persist access token to file *to_file*, to be used in the constructor.

Handles password and OAuth-based authorization.

Will throw a *MastodonIllegalArgumentError* if username / password are wrong, scopes are not valid or granted scopes differ from requested.

For OAuth2 documentation, compare https://github.com/doorkeeper-gem/doorkeeper/wiki/Interacting-as-an-OAuth-client-with-Doorkeeper

Returns the access token as a string.

`Mastodon.`**`auth_request_url`**(*client_id=None, redirect_uris='urn:ietf:wg:oauth:2.0:oob', scopes=['read', 'write', 'follow']*)

Returns the url that a client needs to request the grant from the server.

CHAPTER 7

Versioning

Mastodon.py will check if a certain endpoint is available before doing API calls. By default, it checks against the version of Mastodon retrieved on init(), or the version you specified. Mastodon.py can be set (in the constructor) to either check if an endpoint is available at all (this is the default) or to check if the endpoint is available and behaves as in the newest Mastodon version (with regards to parameters as well as return values). Version checking can also be disabled altogether. If a version check fails, Mastodon.py throws a *MastodonVersionError*.

With the following functions, you can make Mastodon.py re-check the server version or explicitly determine if a specific minimum Version is available.

Mastodon.**retrieve_mastodon_version**()
Determine installed mastodon version and set major, minor and patch (not including RC info) accordingly.

Returns the version string, possibly including rc info.

Mastodon.**verify_minimum_version**(*version_str*)
Update version info from server and verify that at least the specified version is present.

Returns True if version requirement is satisfied, False if not.

CHAPTER 8

Reading data: Instances

This function allows you to fetch information associated with the current instance.

`Mastodon.`**`instance`**`()`

Retrieve basic information about the instance, including the URI and administrative contact email.

Does not require authentication.

Returns an *instance dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.2

CHAPTER 9

Reading data: Timelines

This function allows you to access the timelines a logged in user could see, as well as hashtag timelines and the public timeline.

`Mastodon.`**`timeline`**(*timeline='home'*, *max_id=None*, *since_id=None*, *limit=None*)
Fetch statuses, most recent ones first. *timeline* can be 'home', 'local', 'public', 'tag/hashtag' or 'list/id'. See the following functions documentation for what those do. Local hashtag timelines are supported via the *timeline_hashtag()* function.

The default timeline is the "home" timeline.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`timeline_home`**(*max_id=None*, *since_id=None*, *limit=None*)
Fetch the logged-in users home timeline (i.e. followed users and self).

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`timeline_local`**(*max_id=None*, *since_id=None*, *limit=None*)
Fetches the local / instance-wide timeline, not including replies.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`timeline_public`**(*max_id=None*, *since_id=None*, *limit=None*)
Fetches the public / visible-network timeline, not including replies.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`timeline_hashtag`**(*hashtag*, *local=False*, *max_id=None*, *since_id=None*, *limit=None*)
Fetch a timeline of toots with a given hashtag. The hashtag parameter should not contain the leading #.

Set *local* to True to retrieve only instance-local tagged posts.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

Mastodon.**timeline_list**(*id*, *max_id=None*, *since_id=None*, *limit=None*)
Fetches a timeline containing all the toots by users in a given list.

Returns a list of *toot dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

CHAPTER 10

Reading data: Statuses

These functions allow you to get information about single statuses.

`Mastodon.`**`status`**(*id*)
Fetch information about a single toot.

Does not require authentication for publicly visible statuses.

Returns a *toot dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_context`**(*id*)
Fetch information about ancestors and descendants of a toot.

Does not require authentication for publicly visible statuses.

Returns a *context dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

`Mastodon.`**`status_reblogged_by`**(*id*)
Fetch a list of users that have reblogged a status.

Does not require authentication for publicly visible statuses.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

`Mastodon.`**`status_favourited_by`**(*id*)
Fetch a list of users that have favourited a status.

Does not require authentication for publicly visible statuses.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

`Mastodon.`**`status_card`**(*id*)
Fetch a card associated with a status. A card describes an object (such as an external video or link) embedded into a status.

Does not require authentication for publicly visible statuses.

Returns a *card dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

CHAPTER 11

Reading data: Notifications

This function allows you to get information about a users notifications.

`Mastodon.`**`notifications`**(*id=None*, *max_id=None*, *since_id=None*, *limit=None*)

Fetch notifications (mentions, favourites, reblogs, follows) for the logged-in user.

Can be passed an *id* to fetch a single notification.

Returns a list of *notification dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

CHAPTER 12

Reading data: Accounts

These functions allow you to get information about accounts and their relationships.

`Mastodon.`**`account`**(*id*)
Fetch account information by user *id*.

Returns a *user dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

`Mastodon.`**`account_verify_credentials`**()
Fetch logged-in user's account information.

Returns a *user dict* (Starting from 2.1.0, with an additional "source" field).

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

`Mastodon.`**`account_statuses`**(*id*, *only_media=False*, *pinned=False*, *exclude_replies=False*, *max_id=None*, *since_id=None*, *limit=None*)
Fetch statuses by user *id*. Same options as *timeline()* are permitted. Returned toots are from the perspective of the logged-in user, i.e. all statuses visible to the logged-in user (including DMs) are included.

If *only_media* is set, return only statuses with media attachments. If *pinned* is set, return only statuses that have been pinned. Note that as of Mastodon 2.1.0, this only works properly for instance-local users. If *exclude_replies* is set, filter out all statuses that are replies.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`account_following`**(*id*, *max_id=None*, *since_id=None*, *limit=None*)
Fetch users the given user is following.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

`Mastodon.`**`account_followers`**(*id*, *max_id=None*, *since_id=None*, *limit=None*)
Fetch users the given user is followed by.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

`Mastodon.`**`account_relationships`**`(`*`id`*`)`
Fetch relationship (following, followed_by, blocking, follow requested) of the logged in user to a given account. *id* can be a list.

Returns a list of *relationship dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v1.4.0

`Mastodon.`**`account_search`**`(`*`q`*`, `*`limit=None`*`)`
Fetch matching accounts. Will lookup an account remotely if the search term is in the username@domain format and not yet in the database.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

CHAPTER 13

Reading data: Lists

These functions allow you to view information about lists.

`Mastodon.`**`lists`**`()`
Fetch a list of all the Lists by the logged-in user.

Returns a list of *list dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list`**`(`*id*`)`
Fetch info about a specific list.

Returns a *list dict*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list_accounts`**`(`*id*, *max_id=None*, *since_id=None*, *limit=None*`)`
Get the accounts that are on the given list. A *limit* of 0 can be specified to get all accounts without pagination.

Returns a list of *user dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

CHAPTER 14

Reading data: Follows

`Mastodon.`**`follows`**(*uri*)

Follow a remote user by uri (username@domain).

Returns a *user dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

CHAPTER 15

Reading data: Favourites

Mastodon.**favourites**(*max_id=None*, *since_id=None*, *limit=None*)

Fetch the logged-in user's favourited statuses.

Returns a list of *toot dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

CHAPTER 16

Reading data: Follow requests

Mastodon.**follow_requests**(*max_id=None*, *since_id=None*, *limit=None*)

Fetch the logged-in user's incoming follow requests.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

CHAPTER 17

Reading data: Searching

Mastodon.**search**(*q*, *resolve=False*)

Fetch matching hashtags, accounts and statuses. Will search federated instances if resolve is True.

Returns a *search result dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.1.0

CHAPTER 18

Reading data: Mutes and blocks

These functions allow you to get information about accounts that are muted or blocked by the logged in user.

Mastodon.**mutes**(*max_id=None*, *since_id=None*, *limit=None*)
Fetch a list of users muted by the logged-in user.

Returns a list of *user dicts*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.1.0

Mastodon.**blocks**(*max_id=None*, *since_id=None*, *limit=None*)
Fetch a list of users blocked by the logged-in user.

Returns a list of *user dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

CHAPTER 19

Reading data: Reports

Mastodon.**reports**()

Fetch a list of reports made by the logged-in user.

Returns a list of *report dicts*.

Warning: According to the official API documentation, this method is to be treated as not finalized as of Mastodon 2.1.0.

Added: Mastodon v1.1.0, last changed: Mastodon v1.1.0

CHAPTER 20

Reading data: Domain blocks

Mastodon.**domain_blocks**(*max_id=None*, *since_id=None*, *limit=None*)

Fetch the logged-in user's blocked domains.

Returns a list of blocked domain URLs (as strings, without protocol specifier).

Added: Mastodon v1.4.0, last changed: Mastodon v1.4.0

CHAPTER 21

Reading data: Emoji

Mastodon.**custom_emojis**()

Fetch the list of custom emoji the instance has installed.

Does not require authentication.

Returns a list of *emoji dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

CHAPTER 22

Writing data: Statuses

These functions allow you to post statuses to Mastodon and to interact with already posted statuses.

Mastodon.**status_post**(*status*, *in_reply_to_id=None*, *media_ids=None*, *sensitive=False*, *visibility=''*, *spoiler_text=None*)

Post a status. Can optionally be in reply to another status and contain up to four pieces of media (Uploaded via *media_post()*). media_ids can also be the *media dicts* returned by *media_post()* - they are unpacked automatically.

The *sensitive* boolean decides whether or not media attached to the post should be marked as sensitive, which hides it by default on the Mastodon web front-end.

The visibility parameter is a string value and matches the visibility option on the /api/v1/status POST API endpoint. It accepts any of: 'direct' - post will be visible only to mentioned users 'private' - post will be visible only to followers 'unlisted' - post will be public but not appear on the public timeline 'public' - post will be public

If not passed in, visibility defaults to match the current account's default-privacy setting (starting with Mastodon version 1.6) or its locked setting - private if the account is locked, public otherwise (for Mastodon versions lower than 1.6).

The *spoiler_text* parameter is a string to be shown as a warning before the text of the status. If no text is passed in, no warning will be displayed.

Returns a *toot dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

Mastodon.**toot**(*status*)

Synonym for *status_post()* that only takes the status text as input.

Usage in production code is not recommended.

Returns a *toot dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

Mastodon.**status_reblog**(*id*)

Reblog a status.

Returns a *toot dict* with a new status that wraps around the reblogged one.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_unreblog`**`(`*`id`*`)`
Un-reblog a status.

Returns a *toot dict* with the status that used to be reblogged.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_favourite`**`(`*`id`*`)`
Favourite a status.

Returns a *toot dict* with the favourited status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_unfavourite`**`(`*`id`*`)`
Un-favourite a status.

Returns a *toot dict* with the un-favourited status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_mute`**`(`*`id`*`)`
Mute notifications for a status.

Returns a *toot dict* with the now muted status

Added: Mastodon v1.4.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_unmute`**`(`*`id`*`)`
Unmute notifications for a status.

Returns a *toot dict* with the status that used to be muted.

Added: Mastodon v1.4.0, last changed: Mastodon v2.0.0

`Mastodon.`**`status_delete`**`(`*`id`*`)`
Delete a status

Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

CHAPTER 23

Writing data: Notifications

These functions allow you to clear all or some notifications.

`Mastodon.`**`notifications_clear`**`()`
: Clear out a users notifications

  Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

`Mastodon.`**`notifications_dismiss`**`(`*id*`)`
: Deletes a single notification

  Added: Mastodon v1.3.0, last changed: Mastodon v1.3.0

CHAPTER 24

Writing data: Accounts

These functions allow you to interact with other accounts: To (un)follow and (un)block.

Mastodon.**account_follow**(*id*)
Follow a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.0.0, last changed: Mastodon v1.4.0

Mastodon.**follows**(*uri*)
Follow a remote user by uri (username@domain).

Returns a *user dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.1.0

Mastodon.**account_unfollow**(*id*)
Unfollow a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.0.0, last changed: Mastodon v1.4.0

Mastodon.**account_block**(*id*)
Block a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.0.0, last changed: Mastodon v1.4.0

Mastodon.**account_unblock**(*id*)
Unblock a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.0.0, last changed: Mastodon v1.4.0

Mastodon.**account_mute**(*id*)
Mute a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.0

`Mastodon.`**`account_unmute`**`(`*`id`*`)`

Unmute a user.

Returns a *relationship dict* containing the updated relationship to the user.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.0

`Mastodon.`**`account_update_credentials`**`(`*`display_name=None`*`, `*`note=None`*`, `*`avatar=None`*`, `*`header=None`*`)`

Update the profile for the currently logged-in user.

'note' is the user's bio.

'avatar' and 'header' are images encoded in base64, prepended by a content-type (for example: '[...]')

Returns the updated *user dict* of the logged-in user.

Added: Mastodon v1.1.1, last changed: Mastodon v2.1.0

CHAPTER 25

Writing data: Lists

These functions allow you to create, maintain and delete lists.

When creating lists, note that (As of Mastodon 2.1.0), a user can only have a maximum of 50 lists.

`Mastodon.`**`list_create`**(*title*)
Create a new list with the given *title*.

Returns the *list dict* of the created list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list_update`**(*id*, *title*)
Update info about a list, where "info" is really the lists *title*.

Returns the *list dict* of the modified list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list_delete`**(*id*)
Delete a list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list_accounts_add`**(*id*, *account_ids*)
Add the account(s) given in *account_ids* to the list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

`Mastodon.`**`list_accounts_delete`**(*id*, *account_ids*)
Remove the account(s) given in *account_ids* from the list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

CHAPTER 26

Writing data: Follow requests

These functions allow you to accept or reject incoming follow requests.

`Mastodon.`**`follow_request_authorize`**`(`*id*`)`
: Accept an incoming follow request.

  Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

`Mastodon.`**`follow_request_reject`**`(`*id*`)`
: Reject an incoming follow request.

  Added: Mastodon v1.0.0, last changed: Mastodon v1.0.0

CHAPTER 27

Writing data: Media

This function allows you to upload media to Mastodon. The returned media IDs (Up to 4 at the same time) can then be used with post_status to attach media to statuses.

`Mastodon.`**`media_post`**(*media_file*, *mime_type=None*, *description=None*)

Post an image. *media_file* can either be image data or a file name. If image data is passed directly, the mime type has to be specified manually, otherwise, it is determined from the file name.

Throws a *MastodonIllegalArgumentError* if the mime type of the passed data or file can not be determined properly.

Returns a *media dict*. This contains the id that can be used in status_post to attach the media file to a toot.

Added: Mastodon v1.0.0, last changed: Mastodon v2.0.0

CHAPTER 28

Writing data: Reports

`Mastodon.`**`report`**(*account_id*, *status_ids*, *comment*)

Report statuses to the instances administrators.

Accepts a list of toot IDs associated with the report, and a comment.

Returns a *report dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v1.1.0

CHAPTER 29

Writing data: Domain blocks

These functions allow you to block and unblock all statuses from a domain for the logged-in user.

`Mastodon.`**`domain_block`**`(`*domain=None*`)`
: Add a block for all statuses originating from the specified domain for the logged-in user.

  Added: Mastodon v1.4.0, last changed: Mastodon v1.4.0

`Mastodon.`**`domain_unblock`**`(`*domain=None*`)`
: Remove a domain block for the logged-in user.

  Added: Mastodon v1.4.0, last changed: Mastodon v1.4.0

CHAPTER 30

Pagination

These functions allow for convenient retrieval of paginated data.

Mastodon.**fetch_next**(*previous_page*)
: Fetches the next page of results of a paginated request. Pass in the previous page in its entirety, or the pagination information dict returned as a part of that pages last status ('_pagination_next').

  Returns the next page or None if no further data is available.

Mastodon.**fetch_previous**(*next_page*)
: Fetches the previous page of results of a paginated request. Pass in the previous page in its entirety, or the pagination information dict returned as a part of that pages first status ('_pagination_prev').

  Returns the previous page or None if no further data is available.

Mastodon.**fetch_remaining**(*first_page*)
: Fetches all the remaining pages of a paginated request starting from a first page and returns the entire set of results (including the first page that was passed in) as a big list.

  Be careful, as this might generate a lot of requests, depending on what you are fetching, and might cause you to run into rate limits very quickly.

CHAPTER 31

Streaming

These functions allow access to the streaming API.

If async is False, these methods block forever (or until an exception is raised).

If async is True, the listener will listen on another thread and these methods will return a handle corresponding to the open connection. The connection may be closed at any time by calling the handles close() method, and the status of the connection can be verified calling is_alive() on the handle.

The streaming functions take instances of *StreamListener* as the *listener* parameter. A *CallbackStreamListener* class that allows you to specify function callbacks directly is included for convenience.

`Mastodon.`**`stream_user`**(*listener*, *async=False*)
Streams events that are relevant to the authorized user, i.e. home timeline and notifications.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.2

`Mastodon.`**`stream_public`**(*listener*, *async=False*)
Streams public events.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.2

`Mastodon.`**`stream_local`**(*listener*, *async=False*)
Streams local public events.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.2

`Mastodon.`**`stream_hashtag`**(*tag*, *listener*, *async=False*)
Stream for all public statuses for the hashtag 'tag' seen by the connected instance.

Added: Mastodon v1.1.0, last changed: Mastodon v1.4.2

`Mastodon.`**`stream_list`**(*id*, *listener*, *async=False*)
Stream events for the current user, restricted to accounts on the given list.

Added: Mastodon v2.1.0, last changed: Mastodon v2.1.0

31.1 StreamListener

class mastodon.**StreamListener**
Callbacks for the streaming API. Create a subclass, override the on_xxx methods for the kinds of events you're interested in, then pass an instance of your subclass to Mastodon.user_stream(), Mastodon.public_stream(), or Mastodon.hashtag_stream().

StreamListener.**on_update**(*status*)
A new status has appeared! 'status' is the parsed JSON dictionary describing the status.

StreamListener.**on_notification**(*notification*)
A new notification. 'notification' is the parsed JSON dictionary describing the notification.

StreamListener.**on_delete**(*status_id*)
A status has been deleted. status_id is the status' integer ID.

StreamListener.**handle_heartbeat**()
The server has sent us a keep-alive message. This callback may be useful to carry out periodic housekeeping tasks, or just to confirm that the connection is still open.

31.2 CallbackStreamListener

class mastodon.**CallbackStreamListener**(*update_handler=None*, *local_update_handler=None*, *delete_handler=None*, *notification_handler=None*)
Simple callback stream handler class. Can optionally additionally send local update events to a separate handler.

CHAPTER 32

Acknowledgements

Mastodon.py contains work by a large amount of contributors, many of which have put significant work into making it a better library. You can find some information about who helped with which particular feature or fix in the changelog.

Python Module Index

m

Index

Symbols

A

B

C

D

F

H

I

L

M

N

O