Mastodon.py Documentation

Release 1.8.1

Lorenz Diener

Apr 23, 2023

Introduction

CHAPTER 1

Usage

Register your app! This only needs to be done once (per server, or when distributing rather than hosting an application, most likely per device and server). Uncomment the code and substitute in your information:

```
from mastodon import Mastodon

'''
Mastodon.create_app(
    'pytooterapp',
    api_base_url = 'https://mastodon.social',
    to_file = 'pytooter_clientcred.secret'
)
'''
```

Then, log in. This can be done every time your application starts (e.g. when writing a simple bot), or you can use the persisted information:

```
from mastodon import Mastodon

mastodon = Mastodon(client_id = 'pytooter_clientcred.secret',)
mastodon.log_in(
    'my_login_email@example.com',
    'incrediblygoodpassword',
    to_file = 'pytooter_usercred.secret'
)
```

Note that this won't work when using 2FA - you'll have to use OAuth, in that case. To post, create an actual API instance:

```
from mastodon import Mastodon

mastodon = Mastodon(access_token = 'pytooter_usercred.secret')
mastodon.toot('Tooting from Python using #mastodonpy !')
```

CHAPTER 2

Introduction

Mastodon is an ActivityPub-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.

Mastodon.py aims to implement the complete public Mastodon API. As of this time, it is feature complete for Mastodon version 3.5.5. The Mastodon compatible API layers of various other pieces of software as well as forks, while not an official target, should also be basically compatible, and Mastodon.py does make some allowances for behaviour that isn't strictly like that of Mastodon, and attempts to support extensions to the API.

Some usage examples (not neccesarily following app development best practices, but enough to get you started if you learn best by example) can be found at https://github.com/halcy/MastodonpyExamples

CHAPTER 3

Acknowledgements

Mastodon.py contains work by a large number 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.

3.1 General information

3.1.1 Rate limiting

Mastodon's 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).

Note: Rate limit information is available on the *Mastodon* object for applications that implement their own rate limit handling.

`Mastodon.`**`ratelimit_remaining`**
: Number of requests allowed until the next reset.

`Mastodon.`**`ratelimit_reset`**
: Time at which the rate limit will next be reset, as a POSIX timestamp.

`Mastodon.`**`ratelimit_limit`**
: Total number of requests allowed between resets. Typically 300.

`Mastodon.`**`ratelimit_lastcall`**
: Time at which these values have last been seen and updated, as a POSIX timestamp.

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.

3.1.2 Pagination

Many of Mastodon's 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 four parameters: *since_id*, *max_id*, *min_id* and *limit*. *since_id* allows you to specify the smallest id you want in the returned data, but you will still always get the newest data, so if there are too many statuses between the newest one and *since_id*, some will not be returned. *min_id*, on the other hand, gives you statuses with that minimum id and newer, starting at the given id. *max_id*, similarly, allows you to specify the largest id you want. By specifying either min_id or *max_id* (generally, only one, not both, though specifying both is supported starting with Mastodon version 3.3.0) of them you can go through pages forwards and backwards.

On Mastodon mainline, you can, pass datetime objects as IDs when fetching posts, since the IDs used are Snowflake IDs and dates can be approximately converted to those. This is guaranteed to work on mainline Mastodon servers and very likely to work on all forks, but will **not** work on other servers implementing the API, like Pleroma, Misskey or Gotosocial. You should not use this if you want your application to be universally compatible. It's also relatively coarse-grained.

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 - by default, Mastodon will return no more than 40 statuses and no more than 80 accounts no matter how high you set the limit.

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. They are accessible only via attribute-style access. Note that this means that if you want to persist pagination info with your data, you'll have to take care of that manually (or persist objects, not just dicts).

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. For details, see *fetch_next()*, *fetch_previous()*. and *fetch_remaining()*.

3.1.3 IDs and unpacking

Mastodon's API uses IDs in several places: User IDs, Toot IDs, ...

While debugging, it might be tempting to copy-paste 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.

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.

Snowflake IDs

Some IDs in Mastodon (such as those for statuses) are Snowflake IDs. These broadly correspond to times, with a low resolution, so it is possible to convert a time to a Snowflake ID and search for posts between two dates. Mastodon.py will do the conversion for you automatically when you pass a *datetime* object as the id.

Note that this functionality will *not* work on anything but Mastodon and forks, and that it is somewhat inexact due to the relatively low resolution.

3.1.4 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*.

Some functions need to check what version of Mastodon they are talking to. These will generally use a cached version to avoid sending a lot of pointless requests.

Many non-mainline forks have various different formats for their versions and they have different, incompatible ideas about how to report version. Mastodon.py tries its best to figure out what is going on, but success is not guaranteed.

With the following functions, you can make Mastodon.py re-check the server version or explicitly determine if a specific minimum Version is available. Long-running applications that aim to support multiple Mastodon versions should do this from time to time in case a server they are running against updated.

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*, *cached=False*)
: Update version info from server and verify that at least the specified version is present.

  If you specify "cached", the version info update part is skipped.

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

3.1.5 A brief note on block lists

Mastodon.py used to block three instances because these were particularly notorious for harassing trans people and I don't feel like I have an obligation to let software I distribute help people who want my friends to die. I don't want to be associated with that, at all.

Those instances are now all gone, any point that could have been has been made, and there is no list anymore.

Note: Trans rights are human rights.

3.2 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. The class used for this is exposed as *AttribAccessDict*.

Currently, some of these may be out of date - refer to the Mastodon documentation at https://docs.joinmastodon.org/entities/ for when fields seem to be missing. This will be addressed in the next version of Mastodon.py.

3.2.1 User / account 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
    'discoverable': # True if the user is listed in the user directory, false if not.␣
↪None
                    # for remote users.
    'group': # A boolean indicating whether the account represents a group rather␣
↪than an
             # individual.
    '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; for example '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, a user dict of the account this user has
```

(continues on next page)

(continued from previous page)

```
                        # set up as their moved-to address.
    'bot': # Boolean indicating whether this account is automated.
    'fields': # List of up to four dicts with free-form 'name' and 'value' profile␣
↪info.
              # For fields with "this is me" type verification, verified_at is set to␣
↪the
              # last verification date (It is None otherwise)
    'emojis': # List of custom emoji used in name, bio or fields
    'discoverable': # Indicates whether or not a user is visible on the discovery page
}

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

3.2.2 Toot / Status 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
    'bookmarked': # True if the status is bookmarked by the logged in user, False if␣
↪not.
    '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).
```

(continues on next page)

(continued from previous page)

```
    'language': # The language of the toot, if specified by the server,
                # as ISO 639-1 (two-letter) language code.
    'muted': # Boolean denoting whether the user has muted this status by
             # way of conversation muting
    'pinned': # Boolean denoting whether or not the status is currently pinned for the
              # associated account.
    'replies_count': # The number of replies to this status.
    'card': # A preview card for links from the status, if present at time of␣
↪delivery,
            # as card dict.
    'poll': # A poll dict if a poll is attached to this status.
}
```

3.2.3 Status edit dicts

```
mastodonstatus_history(id)[0]
# Returns the following dictionary
{
    'content': # Content for this version of the status
    'spoiler_text': # CW / Spoiler text for this version of the status
    'sensitive': # Whether media in this version of the status is marked as sensitive
    'created_at': # Time at which this version of the status was posted
    'account': # Account dict of the user that posted the status
    'media_attachments': # List of media dicts with the attached media for this␣
↪version of the status
    'emojis'# List of emoji dicts for this version of the status
}
```

3.2.4 Mention dicts

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

3.2.5 Scheduled status / toot dicts

```
mastodon.status_post("text", scheduled_at=the_future)
# Returns the following dictionary:
{
    'id': # Scheduled toot ID (note: Not the id of the toot once it gets posted!)
    'scheduled_at': # datetime object describing when the toot is to be posted
    'params': # Parameters for the scheduled toot, specifically
    {
        'text': # Toot text
        'in_reply_to_id': # ID of the toot this one is a reply to
        'media_ids': # IDs of media attached to this toot
        'sensitive': # Whether this toot is sensitive or not
```

(continues on next page)

(continued from previous page)

```
        'visibility': # Visibility of the toot
        'idempotency': # Idempotency key for the scheduled toot
        'scheduled_at': # Present, but generally "None"
        'spoiler_text': # CW text for this toot
        'application_id': # ID of the application that scheduled the toot
        'poll': # Poll parameters, as a poll dict
    },
    'media_attachments': # Array of media dicts for the attachments to the scheduled␣
↪toot
}
```

3.2.6 Poll dicts

```
# Returns the following dictionary:
mastodon.poll(id)
{
    'id': # The polls ID
    'expires_at': # The time at which the poll is set to expire
    'expired': # Boolean denoting whether you can still vote in this poll
    'multiple': # Boolean indicating whether it is allowed to vote for more than one␣
↪option
    'votes_count': # Total number of votes cast in this poll
    'voted': # Boolean indicating whether the logged-in user has already voted in␣
↪this poll
    'options': # The poll options as a list of dicts, each option with a title and a
               # votes_count field. votes_count can be None if the poll creator has
               # chosen to hide vote totals until the poll expires and it hasn't yet.
    'emojis': # List of emoji dicts for all emoji used in answer strings,
    'own_votes': # The logged-in users votes, as a list of indices to the options.
}
```

3.2.7 Conversation dicts

```
mastodon.conversations()[0]
# Returns the following dictionary:
{
    'id': # The ID of this conversation object
    'unread': # Boolean indicating whether this conversation has yet to be
              # read by the user
    'accounts': # List of accounts (other than the logged-in account) that
                # are part of this conversation
    'last_status': # The newest status in this conversation
}
```

3.2.8 Hashtag dicts

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

(continues on next page)

(continued from previous page)

```
    'history': # List of usage history dicts for up to 7 days. Not present in␣
↪statuses.
}
```

3.2.9 Hashtag usage history dicts

```
{
    'day': # Date of the day this history dict is for
    'uses': # Number of statuses using this hashtag on that day
    'accounts': # Number of accounts using this hashtag in at least one status on␣
↪that day
}
```

3.2.10 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
    'visible_in_picker': # True if the emoji is enabled, False if not.
    'category': # The category to display the emoji under (not present if none is set)
}
```

3.2.11 Application dicts

```
{
    'name': # The applications name
    'website': # The applications website
    'vapid_key': # A vapid key that can be used in web applications
}
```

3.2.12 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
    'blocked_by': # Boolean denoting whether the logged-in user has been blocked by␣
↪the specified user, if information is available
    'muting': # Boolean denoting whether the logged-in user has muted the specified␣
↪user
```

(continues on next page)

(continued from previous page)

```
    'muting_notifications': # Boolean denoting wheter the logged-in user has muted␣
↪notifications
                            # related to 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
    'showing_reblogs': # Boolean denoting whether the specified users reblogs show up␣
↪on the
                       # logged-in users Timeline
    'endorsed': # Boolean denoting wheter the specified user is being endorsed /␣
↪featured by the
                # logged-in user
    'note': # A free text note the logged in user has created for this account (not␣
↪publicly visible)
    'notifying' # Boolean denoting whether the logged-in user has requested to get␣
↪notified every time the followed user posts
}
```

3.2.13 Filter dicts

```
mastodon.filter(<numerical id>)
# Returns the following dictionary:
{
    'id': # Numerical id of the filter
    'phrase': # Filtered keyword or phrase
    'context': # List of places where the filters are applied ('home', 'notifications
↪', 'public', 'thread')
    'expires_at': # Expiry date for the filter
    'irreversible': # Boolean denoting if this filter is executed server-side
                    # or if it should be ran client-side.
    'whole_word': # Boolean denoting whether this filter can match partial words
}
```

3.2.14 Notification dicts

```
mastodon.notifications()[0]
# Returns the following dictionary:
{
    'id': # id of the notification
    'type': # "mention", "reblog", "favourite", "follow", "poll" or "follow_request"
    '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
}
```

3.2.15 Context dicts

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

3.2.16 List dicts

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

3.2.17 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', 'audio' 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 metadata dicts (see below),
            # 'original' and 'small' (preview). Either may be empty.
            # May additionally contain an "fps" field giving a videos frames per␣
↪second (possibly
            # rounded), and a "length" field giving a videos length in a human-
↪readable format.
            # Note that a video may have an image as preview.
            # May also contain a 'focus' dict and a media 'colors' dict.
    'blurhash': # The blurhash for the image, used for preview / placeholder␣
↪generation
    'description': # If set, the user-provided description for this media.
}

# Metadata dicts (image) - all fields are optional:
{
   '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'
}

# Metadata dicts (video, gifv) - all fields are optional:
{
    'width': # Width of the video in pixels
    'heigh': # Height of the video in pixels
```

(continues on next page)

(continued from previous page)

```
    'frame_rate': # Exact frame rate of the video in frames per second.
                  # Can be an integer fraction (i.e. "20/7")
    'duration': # Duration of the video in seconds
    'bitrate': # Average bit-rate of the video in bytes per second
}

# Metadata dicts (audio) - all fields are optional:
{
    'duration': # Duration of the audio file in seconds
    'bitrate': # Average bit-rate of the audio file in bytes per second
}

# Focus Metadata dict:
{
    'x': # Focus point x coordinate (between -1 and 1)
    'y': # Focus point x coordinate (between -1 and 1)
}

# Media colors dict:
{
    'foreground': # Estimated foreground colour for the attachment thumbnail
    'background': # Estimated background colour for the attachment thumbnail
    'accent': # Estimated accent colour for the attachment thumbnail
}
```

3.2.18 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
    'blurhash': # (optional) Blurhash of the preview image
}
```

3.2.19 Search result dicts

```
mastodon.search("<query>")
# Returns the following dictionary
```

(continues on next page)

(continued from previous page)

```
{
    'accounts': # List of user dicts resulting from the query
    'hashtags': # List of hashtag dicts resulting from the query
    'statuses': # List of toot dicts resulting from the query
}
```

3.2.20 Instance dicts

```
mastodon.instance()
# Returns the following dictionary
{
    'domain': # The instances domain name
    'description': # A brief instance description set by the admin
    'short_description': # An even briefer instance description
    'email': # The admin contact email
    'title': # The instance's title
    'uri': # The instance's URL
    'version': # The instance's Mastodon version
    'urls': # Additional URLs dict, presently only 'streaming_api' with the
            # stream websocket address.
    'stats': # A dictionary containing three stats, user_count (number of local␣
↪users),
             # status_count (number of local statuses) and domain_count (number of␣
↪known
             # instance domains other than this one).
    'contact_account': # User dict of the primary contact for the instance
    'languages': # Array of ISO 639-1 (two-letter) language codes the instance
                 # has chosen to advertise.
    'registrations': # Boolean indication whether registrations on this instance are␣
↪open
                     # (True) or not (False)
    'approval_required': # True if account approval is required when registering,
    'rules': # List of dicts with `id` and `text` fields, one for each server rule␣
↪set by the admin
}
```

3.2.21 Activity dicts

```
mastodon.instance_activity()[0]
# Returns the following dictionary
{
    'week': # Date of the first day of the week the stats were collected for
    'logins': # Number of users that logged in that week
    'registrations': # Number of new users that week
    'statuses': # Number of statuses posted that week
}
```

3.2.22 Report dicts

```
mastodon.admin_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.

    # The following fields are only present in the report dicts returned by␣
↪moderation API:
    'comment': # Text comment submitted with the report
    'created_at': # Time at which this report was created, as a datetime object
    'updated_at': # Last time this report has been updated, as a datetime object
    'account': # User dict of the user that filed this report
    'target_account': # Account that has been reported with this report
    'assigned_account': # If the report as been assigned to an account,
                        # User dict of that account (None if not)
    'action_taken_by_account': # User dict of the account that processed this report
    'statuses': # List of statuses attached to the report, as toot dicts
}
```

3.2.23 Push subscription dicts

```
mastodon.push_subscription()
# Returns the following dictionary
{
    'id': # Numerical id of the push subscription
    'endpoint': # Endpoint URL for the subscription
    'server_key': # Server pubkey used for signature verification
    'alerts': # Subscribed events - dict that may contain keys 'follow',
              # 'favourite', 'reblog' and 'mention', with value True
              # if webpushes have been requested for those events.
}
```

3.2.24 Push notification dicts

```
mastodon.push_subscription_decrypt_push(...)
# Returns the following dictionary
{
    'access_token': # Access token that can be used to access the API as the
                    # notified user
    'body': # Text body of the notification
    'icon': # URL to an icon for the notification
    'notification_id': # ID that can be passed to notification() to get the full
                       # notification object,
    'notification_type': # 'mention', 'reblog', 'follow' or 'favourite'
    'preferred_locale': # The user's preferred locale
    'title': # Title for the notification
}
```

3.2.25 Preference dicts

```
mastodon.preferences()
# Returns the following dictionary
{
    'posting:default:visibility': # The default visibility setting for the user's␣
↪posts,
                                  # as a string
    'posting:default:sensitive': # Boolean indicating whether the user's uploads␣
↪should
                                 # be marked sensitive by default
    'posting:default:language': # The user's default post language, if set (None if␣
↪not)
    'reading:expand:media': # How the user wishes to be shown sensitive media. Can be
                            # 'default' (hide if sensitive), 'hide_all' or 'show_all'
    'reading:expand:spoilers': # Boolean indicating whether the user wishes to expand
                               # content warnings by default
}
```

3.2.26 Featured tag dicts

```
mastodon.featured_tags()[0]
# Returns the following dictionary:
{
    'id': # The featured tags id
    'name': # The featured tags name (without leading #)
    'statuses_count': # Number of publicly visible statuses posted with this hashtag␣
↪that this instance knows about
    'last_status_at': # The last time a public status containing this hashtag was␣
↪added to this instance's database
                      # (can be None if there are none)
}
```

3.2.27 Read marker dicts

```
mastodon.markers_get()["home"]
# Returns the following dictionary:
{
    'last_read_id': # ID of the last read object in the timeline
    'version': # A counter that is incremented whenever the marker is set to a new␣
↪status
    'updated_at': # The time the marker was last set, as a datetime object
}
```

3.2.28 Announcement dicts

```
mastodon.annoucements()[0]
# Returns the following dictionary:
{
    'id': # The annoucements id
    'content': # The contents of the annoucement, as an html string
    'starts_at': # The annoucements start time, as a datetime object. Can be None
    'ends_at': # The annoucements end time, as a datetime object. Can be None
```

(continues on next page)

(continued from previous page)

```
    'all_day': # Boolean indicating whether the annoucement represents an "all day"␣
↪event
    'published_at': # The annoucements publish time, as a datetime object
    'updated_at': # The annoucements last updated time, as a datetime object
    'read': # A boolean indicating whether the logged in user has dismissed the␣
↪annoucement
    'mentions': # Users mentioned in the annoucement, as a list of mention dicts
    'tags': # Hashtags mentioned in the announcement, as a list of hashtag dicts
    'emojis': # Custom emoji used in the annoucement, as a list of emoji dicts
    'reactions': # Reactions to the annoucement, as a list of reaction dicts␣
↪(documented inline here):
    [ {
        'name': # Name of the custom emoji or unicode emoji of the reaction
        'count': # Reaction counter (i.e. number of users who have added this␣
↪reaction)
        'me': # True if the logged-in user has reacted with this emoji, false␣
↪otherwise
        'url': # URL for the custom emoji image
        'static_url': # URL for a never-animated version of the custom emoji image
    } ],
}
```

3.2.29 Familiar follower dicts

```
mastodon.account_familiar_followers(1)[0]
# Returns the following dictionary:
{
    'id': # ID of the account for which the familiar followers are being returned
    'accounts': # List of account dicts of the familiar followers
}
```

3.2.30 Admin account dicts

```
mastodon.admin_account(id)
# Returns the following dictionary
{
    'id': # The users id,
    'username': # The users username, no leading @
    'domain': # The users domain
    'created_at': # The time of account creation
    'email': # For local users, the user's email
    'ip': # For local users, the user's last known IP address
    'role': # 'admin', 'moderator' or None
    'confirmed': # For local users, False if the user has not confirmed their email,␣
↪True otherwise
    'suspended': # Boolean indicating whether the user has been suspended
    'silenced': # Boolean indicating whether the user has been suspended
    'disabled': # For local users, boolean indicating whether the user has had their␣
↪login disabled
    'approved': # For local users, False if the user is pending, True otherwise
    'locale': # For local users, the locale the user has set,
    'invite_request': # If the user requested an invite, the invite request comment␣
↪of that user.
```

(continues on next page)

(continued from previous page)

```
    'invited_by_account_id': # Present if the user was invited by another user and␣
↪set to the inviting users id.
    'account': # The user's account, as a standard user dict
}
```

3.2.31 Admin domain block dicts

3.2.32 Admin measure dicts

```
api.admin_measures(datetime.now() - timedelta(hours=24*5), datetime.now(), active_
↪users=True)
# Returns the following dictionary
{
    'key': # Name of the measure returned
    'unit': # Unit for the measure, if available
    'total': # Value of the measure returned
    'human_value': # Human readable variant of the measure returned
    'data': # A list of dicts with the measure broken down by date, as below
}

# The data dicts:
[
    'date': # Date for this row
    'value': # Value of the measure for this row
}
```

3.2.33 Admin dimension dicts

```
api.admin_dimensions(datetime.now() - timedelta(hours=24*5), datetime.now(),␣
↪languages=True)
# Returns the following dictionary
{
    'key': # Name of the dimension returned
    'data': # A list of data dicts, as below
}

# the data dicts:
{
    'key': # category for this row
    'human_key': # Human readable name for the category for this row, when available
    'value': # Numeric value for the category
},
```

3.2.34 Admin retention dicts

```
api.admin_retention(datetime.now() - timedelta(hours=24*5), datetime.now())
# Returns the following dictionary
{
    'period': # Starting time of the period that the data is being returned for
    'frequency': # Time resolution (day or month) for the returned data
```

(continues on next page)

(continued from previous page)

```
    'data': # List of data dicts, as below
}

# the data dicts:
{
    'date': # Date for this entry
    'rate': # Fraction of users retained
    'value': # Absolute number of users retained
}
```

3.3 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. There is a sub-error of *MastodonNetworkError*, *MastodonReadTimeout*, which is thrown when a streaming API stream times out during reading.

MastodonAPIError is an error returned from the Mastodon instance - the server has decided it can't fulfil your request (i.e. you requested info on a user that does not exist). It is further split into *MastodonNotFoundError* (API returned 404) and *MastodonUnauthorizedError* (API returned 401). Different error codes might exist, but are not currently handled separately.

MastodonMalformedEventError is raised when a streaming API listener receives an invalid event. There have been reports that this can sometimes happen after prolonged operation due to an upstream problem in the requests/urllib libraries.

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

MastodonServerError is raised when the server throws an internal error, likely due to server misconfiguration.

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

3.4 App registration, authentication and preferences

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) and out (revoking the access token you are logged in with). These functions allow you to do those things. Additionally, it is also possible to programmatically register a new user.

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.

3.4.1 App registration and information

static Mastodon.**create_app**(*client_name, scopes=['read', 'write', 'follow', 'push'], redirect_uris=None, website=None, to_file=None, api_base_url=None, request_timeout=300, session=None, user_agent='mastodonpy'*)

Create a new app with given *client_name* and *scopes* (The basic scopes are "read", "write", "follow" and "push" - more granular scopes are available, please refer to Mastodon documentation for which) on the instance given by *api_base_url*.

Specify *redirect_uris* if you want users to be redirected to a certain page after authenticating in an OAuth flow. You can specify multiple URLs by passing a list. Note that if you wish to use OAuth authentication with redirects, the redirect URI must be one of the URLs specified here.

Specify *to_file* to persist your app's info to a file so you can use it in the constructor. Specify *website* to give a website for your app.

Specify *session* with a requests.Session for it to be used instead of the default. This can be used to, amongst other things, adjust proxy or SSL certificate settings.

Specify *user_agent* if you want to use a specific name as *User-Agent* header, otherwise "mastodonpy" will be used.

Presently, app registration is open by default, but this is not guaranteed to be the case for all Mastodon instances in the future.

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

Mastodon.**app_verify_credentials**()

Fetch information about the current application.

Returns an *application dict*.

Added: Mastodon v2.0.0, last changed: Mastodon v2.7.2

3.4.2 Authentication

Mastodon.**__init__**(*client_id=None, client_secret=None, access_token=None, api_base_url=None, debug_requests=False, ratelimit_method='wait', ratelimit_pacefactor=1.1, request_timeout=300, mastodon_version=None, version_check_mode='created', session=None, feature_set='mainline', user_agent='mastodonpy', lang=None*)

Create a new API wrapper instance based on the given *client_secret* and *client_id* on the instance given by *api_base_url*. If you give a *client_id* and it is not a file, you must also give a secret. If you specify an *access_token* then you don't need to specify a *client_id*. It is allowed to specify neither - in this case, you will be restricted to only using endpoints that do not require authentication. If a file is given as *client_id*, client ID, secret and base url are read from that file.

You can also specify an *access_token*, directly or as a file (as written by *log_in()*). If a file is given, Mastodon.py also tries to load the base URL from this file, if present. A client id and secret are not required in this case.

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 avoid hitting 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.

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*.

For fine-tuned control over the requests object use *session* with a requests.Session.

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 endpoint's behaviour has changed after the version of Mastodon that is connected has been released. If it is set to "none", version checking is disabled.

feature_set can be used to enable behaviour specific to non-mainline Mastodon API implementations. Details are documented in the functions that provide such functionality. Currently supported feature sets are *mainline*, *fedibird* and *pleroma*.

For some Mastodon instances a *User-Agent* header is needed. This can be set by parameter *user_agent*. Starting from Mastodon.py 1.5.2 *create_app()* stores the application name into the client secret file. If *client_id* points to this file, the app name will be used as *User-Agent* header as default. It is possible to modify old secret files and append a client app name to use it as a *User-Agent* name.

lang can be used to change the locale Mastodon will use to generate responses. Valid parameters are all ISO 639-1 (two letter) or for a language that has none, 639-3 (three letter) language codes. This affects some error messages (those related to validation) and trends. You can change the language using set_language().

If no other *User-Agent* is specified, "mastodonpy" will be used.

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', 'push'], to_file=None*)

Get the access token for a user.

The username is the email address 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 the OAuth flow data or the username / password credentials given are incorrect, and *MastodonAPIError* if all of the requested scopes were not granted.

For OAuth 2, obtain a code via having your user go to the URL returned by *auth_request_url()* and pass it as the code parameter. In this case, make sure to also pass the same redirect_uri parameter as you used when generating the auth request URL.

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', 'push'], force_login=False, state=None, lang=None*)

Returns the URL that a client needs to request an OAuth grant from the server.

To log in with OAuth, send your user to this URL. The user will then log in and get a code which you can pass to *log_in()*.

scopes are as in *log_in()*, redirect_uris is where the user should be redirected to after authentication. Note that *redirect_uris* must be one of the URLs given during app registration. When using urn:ietf:wg:oauth:2.0:oob, the code is simply displayed, otherwise it is added to the given URL as the "code" request parameter.

Pass force_login if you want the user to always log in even when already logged into web Mastodon (i.e. when registering multiple different accounts in an app).

state is the oauth *state* parameter to pass to the server. It is strongly suggested to use a random, nonguessable value (i.e. nothing meaningful and no incrementing ID) to preserve security guarantees. It can be left out for non-web login flows.

Pass an ISO 639-1 (two letter) or, for languages that do not have one, 639-3 (three letter) language code as *lang* to control the display language for the oauth form.

Mastodon.**set_language**(*lang*)
Set the locale Mastodon will use to generate responses. Valid parameters are all ISO 639-1 (two letter) or, for languages that do not have one, 639-3 (three letter) language codes. This affects some error messages (those related to validation) and trends.

Mastodon.**revoke_access_token**()
Revoke the oauth token the user is currently authenticated with, effectively removing the apps access and requiring the user to log in again.

Mastodon.**create_account**(*username, password, email, agreement=False, reason=None, locale='en', scopes=['read', 'write', 'follow', 'push'], to_file=None, return_detailed_error=False*)
Creates a new user account with the given username, password and email. "agreement" must be set to true (after showing the user the instance's user agreement and having them agree to it), "locale" specifies the language for the confirmation email as an ISO 639-1 (two letter) or, if a language does not have one, 639-3 (three letter) language code. *reason* can be used to specify why a user would like to join if approved-registrations mode is on.

Does not require an access token, but does require a client grant.

By default, this method is rate-limited by IP to 5 requests per 30 minutes.

Returns an access token (just like log_in), which it can also persist to to_file, and sets it internally so that the user is now logged in. Note that this token can only be used after the user has confirmed their email.

By default, the function will throw if the account could not be created. Alternately, when *return_detailed_error* is passed, Mastodon.py will return the detailed error response that the API provides (Starting from version 3.4.0 - not checked here) as an dict with error details as the second return value and the token returned as *None* in case of error. The dict will contain a text *error* values as well as a *details* value which is a dict with one optional key for each potential field (*username*, *password*, *email* and *agreement*), each if present containing a dict with an *error* category and free text *description*. Valid error categories are:

- ERR_BLOCKED - When e-mail provider is not allowed
- ERR_UNREACHABLE - When e-mail address does not resolve to any IP via DNS (MX, A, AAAA)
- ERR_TAKEN - When username or e-mail are already taken
- ERR_RESERVED - When a username is reserved, e.g. "webmaster" or "admin"
- ERR_ACCEPTED - When agreement has not been accepted
- ERR_BLANK - When a required attribute is blank
- ERR_INVALID - When an attribute is malformed, e.g. wrong characters or invalid e-mail address
- ERR_TOO_LONG - When an attribute is over the character limit
- ERR_TOO_SHORT - When an attribute is under the character requirement
- ERR_INCLUSION - When an attribute is not one of the allowed values, e.g. unsupported locale

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**email_resend_confirmation**()
Requests a re-send of the users confirmation mail for an unconfirmed logged in user.

Only available to the app that the user originally signed up with.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

3.4.3 User preferences

Mastodon.**preferences**()
Fetch the user's preferences, which can be used to set some default options. As of 2.8.0, apps can only fetch, not update preferences.

Returns a *preference dict*.

Added: Mastodon v2.8.0, last changed: Mastodon v2.8.0

3.5 Statuses, media and polls

3.5.1 Statuses

These functions allow you to get information about single statuses and to post and update them, as well as to favourite, bookmark, mute reblog ("boost") and to undo all of those. For status pinning, check out TODO and TODO on the accounts page.

Reading

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

Does not require authentication for publicly visible statuses.

Returns a *status 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 *account 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 *account 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.

This function is deprecated as of 3.0.0 and the endpoint does not exist anymore - you should just use the "card" field of the status dicts instead. Mastodon.py will try to mimic the old behaviour, but this is somewhat inefficient and not guaranteed to be the case forever.

Returns a *card dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

`Mastodon.`**`status_history`**(*id*)
Returns the edit history of a status as a list of status edit dicts, starting from the original form. Note that this means that a status that has been edited once will have *two* entries in this list, a status that has been edited twice will have three, and so on.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`status_source`**(*id*)
Returns the source of a status for editing.

Return value is a dictionary containing exactly the parameters you could pass to *status_update()* to change nothing about the status, except *status* is *text* instead.

`Mastodon.`**`favourites`**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetch the logged-in user's favourited statuses.

This endpoint uses internal ids for pagination, passing status ids to *max_id*, *min_id*, or *since_id* will not work. Pagination functions *fetch_next()* and *fetch_previous()* should be used instead.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

`Mastodon.`**`bookmarks`**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Get a list of statuses bookmarked by the logged-in user.

This endpoint uses internal ids for pagination, passing status ids to *max_id*, *min_id*, or *since_id* will not work. Pagination functions *fetch_next()* and *fetch_previous()* should be used instead.

Returns a list of *status dicts*.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Writing

`Mastodon.`**`status_post`**(*status*, *in_reply_to_id=None*, *media_ids=None*, *sensitive=False*, *visibility=None*, *spoiler_text=None*, *language=None*, *idempotency_key=None*, *content_type=None*, *scheduled_at=None*, *poll=None*, *quote_id=None*)
Post a status. Can optionally be in reply to another status and contain media.

media_ids should be a list. (If it's not, the function will turn it into one.) It can 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 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.

Specify *language* to override automatic language detection. The parameter accepts all valid ISO 639-1 (2-letter) or for languages where that do not have one, 639-3 (three letter) language codes.

You can set *idempotency_key* to a value to uniquely identify an attempt at posting a status. Even if you call this function more than once, if you call it with the same *idempotency_key*, only one status will be created.

Pass a datetime as *scheduled_at* to schedule the toot for a specific time (the time must be at least 5 minutes into the future). If this is passed, status_post returns a *scheduled status dict* instead.

Pass *poll* to attach a poll to the status. An appropriate object can be constructed using *make_poll()* . Note that as of Mastodon version 2.8.2, you can only have either media or a poll attached, not both at the same time.

Specific to "pleroma" feature set:: Specify *content_type* to set the content type of your post on Pleroma. It accepts 'text/plain' (default), 'text/markdown', 'text/html' and 'text/bbcode'. This parameter is not supported on Mastodon servers, but will be safely ignored if set.

Specific to "fedibird" feature set:: The *quote_id* parameter is a non-standard extension that specifies the id of a quoted status.

Returns a *status dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

Mastodon.**status_reply**(*to_status*, *status*, *in_reply_to_id=None*, *media_ids=None*, *sensitive=False*, *visibility=None*, *spoiler_text=None*, *language=None*, *idempotency_key=None*, *content_type=None*, *scheduled_at=None*, *poll=None*, *untag=False*)

Helper function - acts like status_post, but prepends the name of all the users that are being replied to the status text and retains CW and visibility if not explicitly overridden.

Note that *to_status* should be a *status dict* and not an ID.

Set *untag* to True if you want the reply to only go to the user you are replying to, removing every other mentioned user from the conversation.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

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

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

Usage in production code is not recommended.

Returns a *status dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

Mastodon.**make_poll**(*options*, *expires_in*, *multiple=False*, *hide_totals=False*)

Generate a poll object that can be passed as the *poll* option when posting a status.

options is an array of strings with the poll options (Maximum, by default: 4), expires_in is the time in seconds for which the poll should be open. Set multiple to True to allow people to choose more than one answer. Set hide_totals to True to hide the results of the poll until it has expired.

Mastodon.**status_reblog**(*id*, *visibility=None*)

Reblog / boost a status.

The visibility parameter functions the same as in *status_post()* and allows you to reduce the visibility of a reblogged status.

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

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

Mastodon.**status_bookmark**(*id*)
Bookmark a status as the logged-in user.

Returns a *status dict* with the now bookmarked status

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**status_unbookmark**(*id*)
Unbookmark a bookmarked status for the logged-in user.

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

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

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

Returns the now-deleted status, with an added "source" attribute that contains the text that was used to compose this status (this can be used to power "delete and redraft" functionality)

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

Mastodon.**status_update**(*id*, *status=None*, *spoiler_text=None*, *sensitive=None*, *media_ids=None*, *poll=None*)
Edit a status. The meanings of the fields are largely the same as in *status_post()*, though not every field can be edited.

Note that editing a poll will reset the votes.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

3.5.2 Scheduled statuses

These functions allow you to get information about scheduled statuses and to update scheduled statuses that already exist. To create new scheduled statuses, use *status_post()* with the *scheduled_at* parameter.

Reading

Mastodon.**scheduled_statuses**()
Fetch a list of scheduled statuses

Returns a list of *scheduled status dicts*.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**scheduled_status**(*id*)
Fetch information about the scheduled status with the given id.

Returns a *scheduled status dict*.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Writing

Mastodon.**scheduled_status_update**(*id*, *scheduled_at*)
Update the scheduled time of a scheduled status.

New time must be at least 5 minutes into the future.

Returns a *scheduled status dict*

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**scheduled_status_delete**(*id*)
Deletes a scheduled status.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

3.5.3 Media

This function allows you to upload media to Mastodon and update media uploads. The returned media IDs (Up to 4 at the same time on a default configuration Mastodon instance) can then be used with post_status to attach media to statuses.

Mastodon.**media_post**(*media_file*, *mime_type=None*, *description=None*, *focus=None*, *file_name=None*, *thumbnail=None*, *thumbnail_mime_type=None*, *synchronous=False*)
Post an image, video or audio file. *media_file* can either be data or a file name. If data is passed directly, the mime type has to be specified manually, otherwise, it is determined from the file name. *focus* should be a tuple of floats between -1 and 1, giving the x and y coordinates of the images focus point for cropping (with the origin being the images center).

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

file_name can be specified to upload a file with the given name, which is ignored by Mastodon, but some other Fediverse server software will display it. If no name is specified, a random name will be generated. The filename of a file specified in media_file will be ignored.

Starting with Mastodon 3.2.0, *thumbnail* can be specified in the same way as *media_file* to upload a custom thumbnail image for audio and video files.

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

When using the v2 API (post Mastodon version 3.1.4), the *url* in the returned dict will be *null*, since attachments are processed asynchronously. You can fetch an updated dict using *media*. Pass "synchronous" to emulate the old behaviour. Not recommended, inefficient and deprecated, will eat your API quota, you know the deal.

Added: Mastodon v1.0.0, last changed: Mastodon v3.2.0

`Mastodon.`**`media_update`**(*id*, *description=None*, *focus=None*, *thumbnail=None*, *thumbnail_mime_type=None*)

Update the metadata of the media file with the given *id*. *description* and *focus* and *thumbnail* are as in *media_post()* .

Returns the updated *media dict*.

Added: Mastodon v2.3.0, last changed: Mastodon v3.2.0

3.5.4 Polls

This function allows you to get and refresh information about polls as well as to vote in polls

Reading

`Mastodon.`**`poll`**(*id*)

Fetch information about the poll with the given id

Returns a *poll dict*.

Added: Mastodon v2.8.0, last changed: Mastodon v2.8.0

Writing

`Mastodon.`**`poll_vote`**(*id*, *choices*)

Vote in the given poll.

choices is the index of the choice you wish to register a vote for (i.e. its index in the corresponding polls *options* field. In case of a poll that allows selection of more than one option, a list of indices can be passed.

You can only submit choices for any given poll once in case of single-option polls, or only once per option in case of multi-option polls.

Returns the updated *poll dict*

Added: Mastodon v2.8.0, last changed: Mastodon v2.8.0

3.6 Accounts, relationships and lists

3.6.1 Accounts

These functions allow you to get information about accounts and associated data as well as update that data - profile data (incuding pinned statuses and endorsements) for the logged in users account, and notes for everyone else

Reading

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

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

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

Mastodon.**me**()
Get this user's account. Synonym for *account_verify_credentials()*, does exactly the same thing, just exists because *account_verify_credentials()* has a confusing name.

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

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

Does not require authentication for publicly visible accounts.

Returns a *account dict*.

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

Mastodon.**account_search**(*q*, *limit=None*, *following=False*, *resolve=False*)
Fetch matching accounts. Will lookup an account remotely if the search term is in the username@domain format and not yet in the database. Set *following* to True to limit the search to users the logged-in user follows.

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.3.0

Mastodon.**account_lookup**(*acct*)
Look up an account from user@instance form (@instance allowed but not required for local accounts). Will only return accounts that the instance already knows about, and not do any webfinger requests. Use *account_search* if you need to resolve users through webfinger from remote.

Returns an *account dict*.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

Mastodon.**featured_tags**()
Return the hashtags the logged-in user has set to be featured on their profile as a list of featured tag dicts.

Returns a list of featured tag dicts.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**featured_tag_suggestions**()
Returns the logged-in user's 10 most commonly-used hashtags.

Returns a list of hashtag dicts.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**account_featured_tags**(*id*)
Get an account's featured hashtags.

Returns a list of hashtag dicts (NOT **'featured tag dicts'_**).

Added: Mastodon v3.3.0, last changed: Mastodon v3.3.0

Mastodon.**endorsements**()
Fetch list of users endorsed by the logged-in user.

Returns a list of *account dicts*.

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

Mastodon.**account_statuses**(*id*, *only_media=False*, *pinned=False*, *exclude_replies=False*, *exclude_reblogs=False*, *tagged=None*, *max_id=None*, *min_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. If *exclude_reblogs* is set, filter out all statuses that are reblogs. If *tagged* is set, return only statuses that are tagged with *tagged*. Only a single tag without a '#' is valid.

Does not require authentication for Mastodon versions after 2.7.0 (returns publicly visible statuses in that case), for publicly visible accounts.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

Mastodon.**account_familiar_followers**(*id*)

Find followers for the account given by id (can be a list) that also follow the logged in account.

Returns a list of familiar follower dicts

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**account_lists**(*id*)

Get all of the logged-in user's lists which the specified user is a member of.

Returns a list of list dicts.

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

Writing

Mastodon.**account_update_credentials**(*display_name=None*, *note=None*, *avatar=None*, *avatar_mime_type=None*, *header=None*, *header_mime_type=None*, *locked=None*, *bot=None*, *discoverable=None*, *fields=None*)

Update the profile for the currently logged-in user.

note is the user's bio.

avatar and 'header' are images. As with media uploads, it is possible to either pass image data and a mime type, or a filename of an image file, for either.

locked specifies whether the user needs to manually approve follow requests.

bot specifies whether the user should be set to a bot.

discoverable specifies whether the user should appear in the user directory.

fields can be a list of up to four name-value pairs (specified as tuples) to appear as semi-structured information in the user's profile.

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

Added: Mastodon v1.1.1, last changed: Mastodon v3.1.0

Mastodon.**account_pin**(*id*)

Pin / endorse a user.

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

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

`Mastodon.`**`account_unpin`**(*id*)
Unpin / un-endorse a user.

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

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

`Mastodon.`**`account_note_set`**(*id*, *comment*)
Set a note (visible to the logged in user only) for the given account.

Returns a *status dict* with the *note* updated.

Added: Mastodon v3.2.0, last changed: Mastodon v3.2.0

`Mastodon.`**`featured_tag_create`**(*name*)
Creates a new featured hashtag displayed on the logged-in user's profile.

Returns a *featured tag dict* with the newly featured tag.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

`Mastodon.`**`featured_tag_delete`**(*id*)
Deletes one of the logged-in user's featured hashtags.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

`Mastodon.`**`status_pin`**(*id*)
Pin a status for the logged-in user.

Returns a *status dict* with the now pinned status

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

`Mastodon.`**`status_unpin`**(*id*)
Unpin a pinned status for the logged-in user.

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

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

3.6.2 Following and followers

These functions allow you to get information about the logged in users followers and users that the logged in users follows as well as follow requests and follow suggestions, and to manage that data - most importantly, follow and unfollow users.

Reading

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

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.**follows**(*uri*)
Follow a remote user with username given in username@domain form.

Returns a *account dict*.

Deprecated - avoid using this. Currently uses a backwards compat implementation that may or may not work properly.

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

Mastodon.**follow_requests**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetch the logged-in user's incoming follow requests.

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**suggestions**()
Fetch follow suggestions for the logged-in user.

Returns a list of *account dicts*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Writing

Mastodon.**account_follow**(*id*, *reblogs=True*, *notify=False*)
Follow a user.

Set *reblogs* to False to hide boosts by the followed user. Set *notify* to True to get a notification every time the followed user posts.

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

Added: Mastodon v1.0.0, last changed: Mastodon v3.3.0

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.**follow_request_authorize**(*id*)
Accept an incoming follow request.

Returns the updated *relationship dict* for the requesting account.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

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

Returns the updated *relationship dict* for the requesting account.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

Mastodon.**suggestion_delete**(*account_id*)
Remove the user with the given *account_id* from the follow suggestions.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

3.6.3 Mutes and blocks

These functions allow you to get information about accounts and domains that are muted or blocked by the logged in user, and to block and mute users and domains

Reading

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

Returns a list of *account dicts*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.6.0

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**domain_blocks**(*max_id=None*, *min_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 v2.6.0

Writing

Mastodon.**account_mute**(*id*, *notifications=True*, *duration=None*)
Mute a user.

Set *notifications* to False to receive notifications even though the user is muted from timelines. Pass a *duration* in seconds to have Mastodon automatically lift the mute after that many seconds.

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

Added: Mastodon v1.1.0, last changed: Mastodon v2.4.3

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_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_remove_from_followers**(*id*)
Remove a user from the logged in users followers (i.e. make them unfollow the logged in user / "softblock" them).

Returns a *relationship dict* reflecting the updated following status.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

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

3.6.4 Lists

These functions allow you to view information about lists as well as to create and update them. By default, the maximum number of lists for a user is 50.

Reading

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*, *min_id=None*, *since_id=None*, *limit=None*)
Get the accounts that are on the given list.

Returns a list of *account dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.6.0

Writing

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

3.7 Reading data: Timelines

These functions allow you to access the timelines a logged in user could see, as well as hashtag timelines and the public (federated) and local timelines. For the public, local and hashtag timelines, access is allowed even when not authenticated if the instance admin has enabled this functionality.

Mastodon.**timeline**(*timeline='home'*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
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.

The default timeline is the "home" timeline.

Specify *only_media* to only get posts with attached media. Specify *local* to only get local statuses, and *remote* to only get remote statuses. Some options are mutually incompatible as dictated by logic.

May or may not require authentication depending on server settings and what is specifically requested.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_home**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
Convenience method: Fetches the logged-in user's home timeline (i.e. followed users and self). Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_local**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*)
Convenience method: Fetches the local / instance-wide timeline, not including replies. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_public**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)

Convenience method: Fetches the public / visible-network / federated timeline, not including replies. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_hashtag**(*hashtag*, *local=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *remote=False*)

Convenience method: Fetch a timeline of toots with a given hashtag. The hashtag parameter should not contain the leading #. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_list**(*id*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)

Convenience method: Fetches a timeline containing all the toots by users in a given list. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v3.1.4

Mastodon.**conversations**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches a user's conversations.

Returns a list of conversation dicts.

Added: Mastodon v2.6.0, last changed: Mastodon v2.6.0

3.8 Instance-wide data and search

3.8.1 Instance information

These functions allow you to fetch information associated with the current instance as well as data from the instance-wide profile directory.

Mastodon.**instance**()

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

Does not require authentication unless locked down by the administrator.

Returns an *instance dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.3.0

Mastodon.**instance_activity**()

Retrieve activity stats about the instance. May be disabled by the instance administrator - throws a MastodonNotFoundError in that case.

Activity is returned for 12 weeks going back from the current week.

Returns a list of activity dicts.

Added: Mastodon v2.1.2, last changed: Mastodon v2.1.2

Mastodon.**instance_peers**()

Retrieve the instances that this instance knows about. May be disabled by the instance administrator - throws a MastodonNotFoundError in that case.

Returns a list of URL strings.

Added: Mastodon v2.1.2, last changed: Mastodon v2.1.2

`Mastodon.`**`instance_health`**`()`
Basic health check. Returns True if healthy, False if not.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

`Mastodon.`**`instance_nodeinfo`**`(`*schema='http://nodeinfo.diaspora.software/ns/schema/2.0'*`)`
Retrieves the instance's nodeinfo information.

For information on what the nodeinfo can contain, see the nodeinfo specification: https://github.com/jhass/nodeinfo . By default, Mastodon.py will try to retrieve the version 2.0 schema nodeinfo.

To override the schema, specify the desired schema with the *schema* parameter.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

`Mastodon.`**`instance_rules`**`()`
Retrieve instance rules.

Returns a list of *id* + *text* dicts, same as the *rules* field in the instance dicts.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

Profile directory

`Mastodon.`**`directory`**`(`*offset=None*, *limit=None*, *order=None*, *local=None*`)`
Fetch the contents of the profile directory, if enabled on the server.

offset how many accounts to skip before returning results. Default 0.

limit how many accounts to load. Default 40.

order **"active" to sort by most recently posted statuses (default) or** "new" to sort by most recently created profiles.

local True to return only local accounts.

Returns a list of *account dicts*.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Emoji

`Mastodon.`**`custom_emojis`**`()`
Fetch the list of custom emoji the instance has installed.

Does not require authentication unless locked down by the administrator.

Returns a list of emoji dicts.

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

3.8.2 Announcements

These functions allow you to fetch announcements, mark annoucements read and modify reactions.

Reading

Mastodon.**announcements**()
Fetch currently active announcements.

Returns a list of announcement dicts.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Writing

Mastodon.**announcement_dismiss**(*id*)
Set the given annoucement to read.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**announcement_reaction_create**(*id*, *reaction*)
Add a reaction to an announcement. *reaction* can either be a unicode emoji or the name of one of the instances custom emoji.

Will throw an API error if the reaction name is not one of the allowed things or when trying to add a reaction that the user has already added (adding a reaction that a different user added is legal and increments the count).

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**announcement_reaction_delete**(*id*, *reaction*)
Remove a reaction to an announcement.

Will throw an API error if the reaction does not exist.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

3.8.3 Trends

These functions, when enabled, allow you to fetch trending tags, statuses and links.

Mastodon.**trending_tags**(*limit=None*, *lang=None*)
Fetch trending-hashtag information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Does not require authentication unless locked down by the administrator.

Important versioning note: This endpoint does not exist for Mastodon versions between 2.8.0 (inclusive) and 3.0.0 (exclusive).

Pass *lang* to override the global locale parameter, which may affect trend ordering.

Returns a list of hashtag dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**trending_statuses**(*limit=None*, *lang=None*)
Fetch trending-status information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Pass *lang* to override the global locale parameter, which may affect trend ordering.

Returns a list of *status dicts*, sorted by the instances's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`trending_links`**(*limit=None*, *lang=None*)
Fetch trending-link information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Returns a list of card dicts, sorted by the instances's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`trends`**(*limit=None*)
Old alias for *trending_tags()*

Deprecated. Please use *trending_tags()* instead.

Added: Mastodon v2.4.3, last changed: Mastodon v3.5.0

3.8.4 Search

These functions allow you to search for users, tags and, when enabled, full text, by default within your own posts and those you have interacted with.

`Mastodon.`**`search`**(*q*, *resolve=True*, *result_type=None*, *account_id=None*, *offset=None*, *min_id=None*, *max_id=None*, *exclude_unreviewed=True*)
Fetch matching hashtags, accounts and statuses. Will perform webfinger lookups if resolve is True. Full-text search is only enabled if the instance supports it, and is restricted to statuses the logged-in user wrote or was mentioned in.

result_type can be one of "accounts", "hashtags" or "statuses", to only search for that type of object.

Specify *account_id* to only get results from the account with that id.

offset, *min_id* and *max_id* can be used to paginate.

exclude_unreviewed can be used to restrict search results for hashtags to only those that have been reviewed by moderators. It is on by default. When using the v1 search API (pre 2.4.1), it is ignored.

Will use search_v1 (no tag dicts in return values) on Mastodon versions before 2.4.1), search_v2 otherwise. Parameters other than resolve are only available on Mastodon 2.8.0 or above - this function will throw a MastodonVersionError if you try to use them on versions before that. Note that the cached version number will be used for this to avoid uneccesary requests.

Returns a *search result dict*, with tags as **'hashtag dicts'_**.

Added: Mastodon v1.1.0, last changed: Mastodon v2.8.0

`Mastodon.`**`search_v2`**(*q*, *resolve=True*, *result_type=None*, *account_id=None*, *offset=None*, *min_id=None*, *max_id=None*, *exclude_unreviewed=True*)
Identical to *search_v1()*, except in that it returns tags as hashtag dicts, has more parameters, and resolves by default.

For more details documentation, please see *search()*

Returns a *search result dict*.

Added: Mastodon v2.4.1, last changed: Mastodon v2.8.0

3.9 Notifications and filtering

3.9.1 Notifications

This function allows you to get information about a user's notifications as well as to clear all or some notifications and to mark conversations as read.

Reading

Mastodon.**notifications**(*id=None*, *account_id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *exclude_types=None*, *types=None*, *mentions_only=None*)

Fetch notifications (mentions, favourites, reblogs, follows) for the logged-in user. Pass *account_id* to get only notifications originating from the given account.

There are different types of notifications:

- *follow* - A user followed the logged in user
- *follow_request* - A user has requested to follow the logged in user (for locked accounts)
- *favourite* - A user favourited a post by the logged in user
- *reblog* - A user reblogged a post by the logged in user
- *mention* - A user mentioned the logged in user
- *poll* - A poll the logged in user created or voted in has ended
- *update* - A status the logged in user has reblogged (and only those, as of 4.0.0) has been edited
- *status* - A user that the logged in user has enabned notifications for has enabled *notify* (see *account_follow()*)
- *admin.sign_up* - For accounts with appropriate permissions (TODO: document which those are when adding the permission API): A new user has signed up
- *admin.report* - For accounts with appropriate permissions (TODO: document which those are when adding the permission API): A new report has been received

Parameters *exclude_types* and *types* are array of these types, specifying them will in- or exclude the types of notifications given. It is legal to give both parameters at the same tine, the result will then be the intersection of the results of both filters. Specifying *mentions_only* is a deprecated way to set *exclude_types* to all but mentions.

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

Returns a list of notification dicts.

Added: Mastodon v1.0.0, last changed: Mastodon v3.5.0

Writing

Mastodon.**notifications_clear**()

Clear out a user's 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 v2.9.2

Mastodon.**conversations_read**(*id*)
Marks a single conversation as read.

Returns the updated *conversation dict*.

Added: Mastodon v2.6.0, last changed: Mastodon v2.6.0

3.9.2 Keyword filters

These functions allow you to get information about keyword filters as well as to create and update filters.

Very Important Note: The filtering system was revised in 4.0.0. This means that these functions will now not work anymore if an instance is on Mastodon 4.0.0 or above. When updating Mastodon.py for 4.0.0, we'll make an effort to emulate old behaviour, but this will not always be possible. Consider these methods deprecated, for now.

Reading

Mastodon.**filters**()
Fetch all of the logged-in user's filters.

Returns a list of filter dicts. Not paginated.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter**(*id*)
Fetches information about the filter with the specified *id*.

Returns a *filter dict*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filters_apply**(*objects*, *filters*, *context*)
Helper function: Applies a list of filters to a list of either statuses or notifications and returns only those matched by none. This function will apply all filters that match the context provided in *context*, i.e. if you want to apply only notification-relevant filters, specify 'notifications'. Valid contexts are 'home', 'notifications', 'public' and 'thread'.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Writing

Mastodon.**filter_create**(*phrase*, *context*, *irreversible=False*, *whole_word=True*, *expires_in=None*)
Creates a new keyword filter. *phrase* is the phrase that should be filtered out, *context* specifies from where to filter the keywords. Valid contexts are 'home', 'notifications', 'public' and 'thread'.

Set *irreversible* to True if you want the filter to just delete statuses server side. This works only for the 'home' and 'notifications' contexts.

Set *whole_word* to False if you want to allow filter matches to start or end within a word, not only at word boundaries.

Set *expires_in* to specify for how many seconds the filter should be kept around.

Returns the *filter dict* of the newly created filter.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter_update**(*id*, *phrase=None*, *context=None*, *irreversible=None*, *whole_word=None*, *expires_in=None*)

Updates the filter with the given *id*. Parameters are the same as in *filter_create()*.

Returns the *filter dict* of the updated filter.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter_delete**(*id*)

Deletes the filter with the given *id*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

3.9.3 Push notifications

Mastodon supports the delivery of notifications via webpush.

These functions allow you to manage webpush subscriptions and to decrypt received pushes. Note that the intended setup is not Mastodon pushing directly to a user's client - the push endpoint should usually be a relay server that then takes care of delivering the (encrypted) push to the end user via some mechanism, where it can then be decrypted and displayed.

Mastodon allows an application to have one webpush subscription per user at a time.

All crypto utilities require Mastodon.py's optional "webpush" feature dependencies (specifically, the "cryptography" and "http_ece" packages).

Mastodon.**push_subscription**()

Fetch the current push subscription the logged-in user has for this app.

Returns a *push subscription dict*.

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

Mastodon.**push_subscription_set**(*endpoint*, *encrypt_params*, *follow_events=None*, *favourite_events=None*, *reblog_events=None*, *mention_events=None*, *poll_events=None*, *follow_request_events=None*, *status_events=None*, *policy='all'*)

Sets up or modifies the push subscription the logged-in user has for this app.

endpoint is the endpoint URL mastodon should call for pushes. Note that mastodon requires https for this URL. *encrypt_params* is a dict with key parameters that allow the server to encrypt data for you: A public key *pubkey* and a shared secret *auth*. You can generate this as well as the corresponding private key using the *push_subscription_generate_keys()* function.

policy controls what sources will generate webpush events. Valid values are *all*, *none*, *follower* and *followed*.

The rest of the parameters controls what kind of events you wish to subscribe to.

Returns a *push subscription dict*.

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

Mastodon.**push_subscription_update**(*follow_events=None*, *favourite_events=None*, *reblog_events=None*, *mention_events=None*, *poll_events=None*, *follow_request_events=None*)

Modifies what kind of events the app wishes to subscribe to.

Returns the updated *push subscription dict*.

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

Mastodon.**push_subscription_generate_keys**()
Generates a private key, public key and shared secret for use in webpush subscriptions.

Returns two dicts: One with the private key and shared secret and another with the public key and shared secret.

Mastodon.**push_subscription_decrypt_push**(*data*, *decrypt_params*, *encryption_header*, *crypto_key_header*)
Decrypts *data* received in a webpush request. Requires the private key dict from *push_subscription_generate_keys()* (*decrypt_params*) as well as the Encryption and server Crypto-Key headers from the received webpush

Returns the decoded webpush as a *push notification dict*.

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

Usage example

This is a minimal usage example for the push API, including a small http server to receive webpush notifications.

```
api = Mastodon(...)
keys = api.push_subscription_generate_keys()
api.push_subscription_set(endpoint, keys[1], mention_events=1)

class Handler(http.server.BaseHTTPRequestHandler):
    def do_POST(self):
        self.send_response(201)
        self.send_header('Location', '')  # Mastodon doesn't seem to care about this
        self.end_headers()
        data = self.rfile.read(int(self.headers['content-length']))
        np = api.push_subscription_decrypt_push(data, keys[0], self.headers[
↪'Encryption'], self.headers['Crypto-Key'])
        n = api.notifications(id=np.notification_id)
        s = n.status
        self.log_message('\nFrom: %s\n%s', s.account.acct, s.content)
httpd = http.server.HTTPServer(('', 42069), Handler)

try:
    httpd.serve_forever()
except KeyboardInterrupt:
    pass
finally:
    httpd.server_close()
    api.push_subscription_delete()
```

3.10 Streaming

These functions allow access to the streaming API. For the public, local and hashtag streams, access is generally possible without authenticating.

If *run_async* is False, these methods block forever (or until an error is encountered).

If *run_async* is True, the listener will listen on another thread and these methods will return a handle corresponding to the open connection. If, in addition, *reconnect_async* is True, the thread will attempt to reconnect to the streaming API if any errors are encountered, waiting *reconnect_async_wait_sec* seconds between reconnection attempts. Note that no effort is made to "catch up" - events created while the connection is broken will not be received. If you need to make sure to get absolutely all notifications / deletes / toots, you will have to do that manually, e.g. using the *on_abort*

handler to fill in events since the last received one and then reconnecting. Both *run_async* and *reconnect_async* default to false, and you'll have to set each to true separately to get the behaviour described above.

The connection may be closed at any time by calling the handles close() method. The current status of the handler thread can be checked with the handles is_alive() function, and the streaming status can be checked by calling is_receiving().

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.

For new well-known events implement the streaming function in *StreamListener* or *CallbackStreamListener*. The function name is *on_* + the event name. If the event name contains dots, they are replaced with underscored, e.g. for an event called 'status.update' the listener function should be named *on_status_update*.

It may be that future Mastodon versions will come with completely new (unknown) event names. If you want to do something when such an event is received, override the listener function *on_unknown_event*. This has an additional parameter *name* which informs about the name of the event. *unknown_event* contains the content of the event. Alternatively, a callback function can be passed in the *unknown_event_handler* parameter in the *CallbackStreamListener* constructor.

Note that the *unknown_event* handler is *not* guaranteed to receive events once they have been implemented. Events will only go to this handler temporarily, while Mastodon.py has not been updated. Changes to what events do and do not go into the handler will not be considered a breaking change. If you want to handle a new event whose name you _do_ know, define an appropriate handler in your StreamListener, which will work even if it is not listed here.

When in not-async mode or async mode without async_reconnect, the stream functions may raise various exceptions: *MastodonMalformedEventError* if a received event cannot be parsed and *MastodonNetworkError* if any connection problems occur.

Mastodon.py currently does not support websocket based, multiplexed streams, but might in the future.

3.10.1 Stream endpoints

`Mastodon.`**`stream_user`**(*listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)

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*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*, *local=False*, *remote=False*)

Streams public events.

Set *local* to True to only get local statuses. Set *remote* to True to only get remote statuses.

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

`Mastodon.`**`stream_local`**(*listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)

Streams local public events.

This function is deprecated. Please use stream_public() with parameter *local* set to True instead.

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

`Mastodon.`**`stream_hashtag`**(*tag*, *listener*, *local=False*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)

Stream for all public statuses for the hashtag 'tag' seen by the connected instance.

Set *local* to True to only get local statuses.

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

Mastodon.**stream_list**(*id*, *listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)
Stream events for the current user, restricted to accounts on the given list.

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

Mastodon.**stream_healthy**()
Returns without True if streaming API is okay, False or raises an error otherwise.

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

3.10.2 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 *status dict* describing the status.

StreamListener.**on_notification**(*notification*)
A new notification. *notification* is the parsed *notification dict* describing the notification.

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

StreamListener.**on_conversation**(*conversation*)
A direct message (in the direct stream) has been received. *conversation* is the parsed *conversation dict* dictionary describing the conversation

StreamListener.**on_status_update**(*status*)
A status has been edited. 'status' is the parsed JSON dictionary describing the updated status.

StreamListener.**on_unknown_event**(*name*, *unknown_event=None*)
An unknown mastodon API event has been received. The name contains the event-name and unknown_event contains the content of the unknown event.

StreamListener.**on_abort**(*err*)
There was a connection error, read timeout or other error fatal to the streaming connection. The exception object about to be raised is passed to this function for reference.

Note that the exception will be raised properly once you return from this function, so if you are using this handler to reconnect, either never return or start a thread and then catch and ignore the exception.

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.

CallbackStreamListener

class mastodon.**CallbackStreamListener**(*update_handler=None*, *local_update_handler=None*, *delete_handler=None*, *notification_handler=None*, *conversation_handler=None*, *unknown_event_handler=None*, *status_update_handler=None*, *filters_changed_handler=None*, *announcement_handler=None*, *announcement_reaction_handler=None*, *announcement_delete_handler=None*, *encrypted_message_handler=None*)

Simple callback stream handler class. Can optionally additionally send local update events to a separate handler. Define an unknown_event_handler for new Mastodon API events. This handler is *not* guaranteed to receive these events forever, and should only be used for diagnostics.

3.11 Misc: Markers, reports

3.11.1 Markers

These functions allow you to interact with the timeline "last read" markers, to allow for persisting where the user was reading a timeline between sessions and clients / devices.

Reading

Mastodon.**markers_get**(*timeline=['home']*)

Get the last-read-location markers for the specified timelines. Valid timelines are the same as in *timeline()*

Note that despite the singular name, *timeline* can be a list.

Returns a dict of read marker dicts, keyed by timeline name.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Writing

Mastodon.**markers_set**(*timelines*, *last_read_ids*)

Set the "last read" marker(s) for the given timeline(s) to the given id(s)

Note that if you give an invalid timeline name, this will silently do nothing.

Returns a dict with the updated read marker dicts, keyed by timeline name.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

3.11.2 Reports

Reading

In Mastodon versions before 2.5.0 this function allowed for the retrieval of reports filed by the logged in user. It has since been removed.

Mastodon.**reports**()
Fetch a list of reports made by the logged-in user.

Returns a list of report dicts.

Warning: This method has now finally been removed, and will not work on Mastodon versions 2.5.0 and above.

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

Writing

This function allows you to report a user to the instance moderators as well as to the users home instance.

Mastodon.**report**(*account_id*, *status_ids=None*, *comment=None*, *forward=False*, *category=None*, *rule_ids=None*)
Report statuses to the instances administrators.

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

Starting with Mastodon 3.5.0, you can also pass a *category* (one out of "spam", "violation" or "other") and *rule_ids* (a list of rule IDs corresponding to the rules returned by the *instance()* API).

Set *forward* to True to forward a report of a remote user to that users instance as well as sending it to the instance local administrators.

Returns a *report dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v3.5.0

3.12 Utility: Pagination and Blurhash

3.12.1 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.

3.12.2 Blurhash decoding

This function allows for easy basic decoding of blurhash strings to images. This requires Mastodon.pys optional "blurhash" feature dependencies.

Mastodon.**decode_blurhash**(*media_dict*, *out_size=(16, 16)*, *size_per_component=True*, *return_linear=True*)

Basic media-dict blurhash decoding.

out_size is the desired result size in pixels, either absolute or per blurhash component (this is the default).

By default, this function will return the image as linear RGB, ready for further scaling operations. If you want to display the image directly, set return_linear to False.

Returns the decoded blurhash image as a three-dimensional list: [height][width][3], with the last dimension being RGB colours.

For further info and tips for advanced usage, refer to the documentation for the blurhash module: https://github.com/halcy/blurhash-python

3.13 Administration and moderation

These functions allow you to perform moderation actions on users and generally process reports using the API. To do this, you need access to the "admin:read" and/or "admin:write" scopes or their more granular variants (both for the application and the access token), as well as at least moderator access. Mastodon.py will not request these by default, as that would be very dangerous.

BIG WARNING: TREAT ANY ACCESS TOKENS THAT HAVE ADMIN CREDENTIALS AS EXTREMELY, MASSIVELY SENSITIVE DATA AND MAKE EXTRA SURE TO REVOKE THEM AFTER TESTING, NOT LET THEM SIT IN FILES SOMEWHERE, TRACK WHICH ARE ACTIVE, ET CETERA. ANY EXPOSURE OF SUCH ACCESS TOKENS MEANS YOU EXPOSE THE PERSONAL DATA OF ALL YOUR USERS TO WHOEVER HAS THESE TOKENS. TREAT THEM WITH EXTREME CARE.

This is not to say that you should not treat access tokens from admin accounts that do not have admin: scopes attached with a lot of care, but be extra careful with those that do.

3.13.1 Accounts

Mastodon.**admin_accounts_v2**(*origin=None*, *by_domain=None*, *status=None*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *permissions=None*, *invited_by=None*, *role_ids=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches a list of accounts that match given criteria. By default, local accounts are returned.

- Set *origin* to "local" or "remote" to get only local or remote accounts.
- Set *by_domain* to a domain to get only accounts from that domain.
- Set *status* to one of "active", "pending", "disabled", "silenced" or "suspended" to get only accounts with that moderation status (default: active)
- Set *username* to a string to get only accounts whose username contains this string.
- Set *display_name* to a string to get only accounts whose display name contains this string.
- Set *email* to an email to get only accounts with that email (this only works on local accounts).
- Set *ip* to an ip (as a string, standard v4/v6 notation) to get only accounts whose last active ip is that ip (this only works on local accounts).
- Set *permissions* to "staff" to only get accounts with staff permissions.
- Set *invited_by* to an account id to get only accounts invited by this user.
- Set *role_ids* to a list of role IDs to get only accounts with those roles.

Returns a list of admin account dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v4.0.0

Mastodon.**admin_accounts**(*remote=False*, *by_domain=None*, *status='active'*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *staff_only=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Currently a synonym for admin_accounts_v1, now deprecated. You are strongly encouraged to use admin_accounts_v2 instead, since this one is kind of bad.

!!!!! This function may be switched to calling the v2 API in the future. This is your warning. If you want to keep using v1, use it explicitly. !!!!!

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_accounts_v1**(*remote=False*, *by_domain=None*, *status='active'*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *staff_only=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches a list of accounts that match given criteria. By default, local accounts are returned.

- Set *remote* to True to get remote accounts, otherwise local accounts are returned (default: local accounts)
- Set *by_domain* to a domain to get only accounts from that domain.
- Set *status* to one of "active", "pending", "disabled", "silenced" or "suspended" to get only accounts with that moderation status (default: active)
- Set *username* to a string to get only accounts whose username contains this string.
- Set *display_name* to a string to get only accounts whose display name contains this string.
- Set *email* to an email to get only accounts with that email (this only works on local accounts).
- Set *ip* to an ip (as a string, standard v4/v6 notation) to get only accounts whose last active ip is that ip (this only works on local accounts).
- Set *staff_only* to True to only get staff accounts (this only works on local accounts).

Note that setting the boolean parameters to False does not mean "give me users to which this does not apply" but instead means "I do not care if users have this attribute".

Deprecated in Mastodon version 3.5.0.

Returns a list of admin account dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account**(*id*)

Fetches a single *admin account dict* for the user with the given id.

Returns that dict.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_enable**(*id*)

Reenables login for a local account for which login has been disabled.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_approve**(*id*)

Approves a pending account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_reject**(*id*)
Rejects and deletes a pending account.

Returns the updated *admin account dict* for the account that is now gone.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_unsilence**(*id*)
Unsilences an account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_unsuspend**(*id*)
Unsuspends an account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_moderate**(*id*, *action=None*, *report_id=None*, *warning_preset_id=None*, *text=None*, *send_email_notification=True*)
Perform a moderation action on an account.

Valid actions are:

- "disable" - for a local user, disable login.
- "silence" - hide the users posts from all public timelines.
- "suspend" - irreversibly delete all the user's posts, past and future.
- "sensitive" - forcce an accounts media visibility to always be sensitive.

If no action is specified, the user is only issued a warning.

Specify the id of a report as *report_id* to close the report with this moderation action as the resolution. Specify *warning_preset_id* to use a warning preset as the notification text to the user, or *text* to specify text directly. If both are specified, they are concatenated (preset first). Note that there is currently no API to retrieve or create warning presets.

Set *send_email_notification* to False to not send the user an email notification informing them of the moderation action.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

3.13.2 Reports

Mastodon.**admin_reports**(*resolved=False*, *account_id=None*, *target_account_id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetches the list of reports.

Set *resolved* to True to search for resolved reports. *account_id* and *target_account_id* can be used to get reports filed by or about a specific user.

Returns a list of report dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report**(*id*)
Fetches the report with the given id.

Returns a *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

`Mastodon.`**`admin_report_assign`**(*id*)
Assigns the given report to the logged-in user.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

`Mastodon.`**`admin_report_unassign`**(*id*)
Unassigns the given report from the logged-in user.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

`Mastodon.`**`admin_report_reopen`**(*id*)
Reopens a closed report.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

`Mastodon.`**`admin_report_resolve`**(*id*)
Marks a report as resolved (without taking any action).

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

3.13.3 Trends

`Mastodon.`**`admin_trending_tags`**(*limit=None*)
Admin version of *trending_tags()*. Includes unapproved tags.

Returns a list of hashtag dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`admin_trending_statuses`**()
Admin version of *trending_statuses()*. Includes unapproved tags.

Returns a list of *status dicts*, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`admin_trending_links`**()
Admin version of *trending_links()*. Includes unapproved tags.

Returns a list of card dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

`Mastodon.`**`admin_domain_blocks`**(*id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetches a list of blocked domains. Requires scope *admin:read:domain_blocks*.

Provide an *id* to fetch a specific domain block based on its database id.

Returns a list of admin domain block dicts, raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

3.13.4 Federation

Mastodon.**admin_create_domain_block**(*domain: str*, *severity: str = None*, *reject_media: bool = None*, *reject_reports: bool = None*, *private_comment: str = None*, *public_comment: str = None*, *obfuscate: bool = None*)

Perform a moderation action on a domain. Requires scope *admin:write:domain_blocks*.

Valid severities are:

- "silence" - hide all posts from federated timelines and do not show notifications to local users from the remote instance's users unless they are following the remote user.
- "suspend" - deny interactions with this instance going forward. This action is reversible.
- "limit" - generally used with reject_media=true to force reject media from an instance without silencing or suspending..

If no action is specified, the domain is only silenced. *domain* is the domain to block. Note that using the top level domain will also imapct all subdomains. ie, example.com will also impact subdomain.example.com. *reject_media* will not download remote media on to your local instance media storage. *reject_reports* ignores all reports from the remote instance. *private_comment* sets a private admin comment for the domain. *public_comment* sets a publicly available comment for this domain, which will be available to local users and may be available to everyone depending on your settings. *obfuscate* censors some part of the domain name. Useful if the domain name contains unwanted words like slurs.

Returns the new domain block as an *admin domain block dict*.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_update_domain_block**(*id*, *severity: str = None*, *reject_media: bool = None*, *reject_reports: bool = None*, *private_comment: str = None*, *public_comment: str = None*, *obfuscate: bool = None*)

Modify existing moderation action on a domain. Requires scope *admin:write:domain_blocks*.

Valid severities are:

- "silence" - hide all posts from federated timelines and do not show notifications to local users from the remote instance's users unless they are following the remote user.
- "suspend" - deny interactions with this instance going forward. This action is reversible.
- "limit" - generally used with reject_media=true to force reject media from an instance without silencing or suspending.

If no action is specified, the domain is only silenced. *domain* is the domain to block. Note that using the top level domain will also imapct all subdomains. ie, example.com will also impact subdomain.example.com. *reject_media* will not download remote media on to your local instance media storage. *reject_reports* ignores all reports from the remote instance. *private_comment* sets a private admin comment for the domain. *public_comment* sets a publicly available comment for this domain, which will be available to local users and may be available to everyone depending on your settings. *obfuscate* censors some part of the domain name. Useful if the domain name contains unwanted words like slurs.

Returns the modified domain block as an *admin domain block dict*, raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_delete_domain_block**(*id=None*)

Removes moderation action against a given domain. Requires scope *admin:write:domain_blocks*.

Provide an *id* to remove a specific domain block based on its database id.

Raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

3.13.5 Moderation actions

Mastodon.**admin_measures**(*start_at*, *end_at*, *active_users=False*, *new_users=False*, *interactions=False*, *opened_reports=False*, *resolved_reports=False*, *tag_accounts=None*, *tag_uses=None*, *tag_servers=None*, *instance_accounts=None*, *instance_media_attachments=None*, *instance_reports=None*, *instance_statuses=None*, *instance_follows=None*, *instance_followers=None*)

Retrieves numerical instance information for the time period (at day granularity) between *start_at* and *end_at*.

- *active_users*: Pass true to retrieve the number of active users on your instance within the time period
- *new_users*: Pass true to retrieve the number of users who joined your instance within the time period
- *interactions*: Pass true to retrieve the number of interactions (favourites, boosts, replies) on local statuses within the time period
- *opened_reports*: Pass true to retrieve the number of reports filed within the time period
- *resolved_reports* = Pass true to retrieve the number of reports resolved within the time period
- *tag_accounts*: Pass a tag ID to get the number of accounts which used that tag in at least one status within the time period
- *tag_uses*: Pass a tag ID to get the number of statuses which used that tag within the time period
- *tag_servers*: Pass a tag ID to to get the number of remote origin servers for statuses which used that tag within the time period
- *instance_accounts*: Pass a domain to get the number of accounts originating from that remote domain within the time period
- *instance_media_attachments*: Pass a domain to get the amount of space used by media attachments from that remote domain within the time period
- *instance_reports*: Pass a domain to get the number of reports filed against accounts from that remote domain within the time period
- *instance_statuses*: Pass a domain to get the number of statuses originating from that remote domain within the time period
- *instance_follows*: Pass a domain to get the number of accounts from a remote domain followed by that local user within the time period
- *instance_followers*: Pass a domain to get the number of local accounts followed by accounts from that remote domain within the time period

This API call is relatively expensive - watch your servers load if you want to get a lot of statistical data. Especially the instance_statuses stats might take a long time to compute and, in fact, time out.

There is currently no way to get tag IDs implemented in Mastodon.py, because the Mastodon public API does not implement one. This will be fixed in a future release.

Returns a list of admin measure dicts.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_dimensions**(*start_at*, *end_at*, *limit=None*, *languages=False*, *sources=False*, *servers=False*, *space_usage=False*, *software_versions=False*, *tag_servers=None*, *tag_languages=None*, *instance_accounts=None*, *instance_languages=None*)

Retrieves primarily categorical instance information for the time period (at day granularity) between *start_at* and *end_at*.

- *languages*: Pass true to get the most-used languages on this server
- *sources*: Pass true to get the most-used client apps on this server
- *servers*: Pass true to get the remote servers with the most statuses
- *space_usage*: Pass true to get the how much space is used by different components your software stack
- *software_versions*: Pass true to get the version numbers for your software stack
- *tag_servers*: Pass a tag ID to get the most-common servers for statuses including a trending tag
- *tag_languages*: Pass a tag ID to get the most-used languages for statuses including a trending tag
- *instance_accounts*: Pass a domain to get the most-followed accounts from a remote server
- *instance_languages*: Pass a domain to get the most-used languages from a remote server

Pass *limit* to set how many results you want on queries where that makes sense.

This API call is relatively expensive - watch your servers load if you want to get a lot of statistical data.

There is currently no way to get tag IDs implemented in Mastodon.py, because the Mastodon public API does not implement one. This will be fixed in a future release.

Returns a list of admin dimension dicts.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_retention**(*start_at*, *end_at*, *frequency='day'*)

Gets user retention statistics (at *frequency* - "day" or "month" - granularity) between *start_at* and *end_at*.

Returns a list of admin retention dicts

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

3.14 Contributing

3.14.1 How to contribute

Mastodon.py is incomplete a lot of the time because Mastodon has a very rich API with many functions, not all of which are implemented here. Even when it is complete for a given Mastodon API version, there are forks and other Mastodon-API-compatible software that implement their own methods which Mastodon.py could in principle support. And even when all of that work is done, it will inevitably have bugs, or places where the library could be made easier to use (which, really, are also bugs), missing tests that could catch bugs quicker, tooling to make updating everything faster, et cetera.

You can help get more of this done, and you should! This can take many forms: If you notice somtehing is missing, broken or confusing:

- You could file an issue on github, either with or without suggestions for how to fix the issue: https://github.com/halcy/Mastodon.py/issues
- You could, after filing an issue, do a PR that fixes that issue

- You could even just vaguely complain in my (https://icosahedron.website/@halcy) general direction on Mastodon

All of these help immensely, even if it's just "hey, I don't really get why X isn't working". We can't make the library better if we don't know what the actual issues people have are, so while I'm not going to implement every suggestion and do have some ideas of what does and does not make a good library, your feedback is, in fact, extremely valuable and welcome.

If you're looking for some "starter issues" to address: Currently, we don't have support for much of any of the new 4.0.0 API endpoints implemented. Pick one and have a go, especially from the admin API. Tests are somewhat annoying to set up, as they need to run against a live mastodon instance - great if you can write them, but feel free to skip out on them, too, or just write them "in the dry" without actually running them and leaving that for someone else.

3.14.2 Tests

Mastodon.py has an extensive suite of tests. The purpose of these is twofold:

- Make sure nothing is broken and that there aren't any regressions
- Where the official docs are unclear, verify assumptions we make about the Mastodon API and document the results

The tests use pytest and pytest-vcr so that they can be ran even without a mastodon server, but new tests require setting up a mastodon dev server. Further documentation can be found in the "tests" directory in the repository.

3.15 Every function on a huge CTRL-F-able page

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*, *cached=False*)
Update version info from server and verify that at least the specified version is present.

If you specify "cached", the version info update part is skipped.

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

static Mastodon.**create_app**(*client_name*, *scopes=['read', 'write', 'follow', 'push']*, *redirect_uris=None*, *website=None*, *to_file=None*, *api_base_url=None*, *request_timeout=300*, *session=None*, *user_agent='mastodonpy'*)
Create a new app with given *client_name* and *scopes* (The basic scopes are "read", "write", "follow" and "push" - more granular scopes are available, please refer to Mastodon documentation for which) on the instance given by *api_base_url*.

Specify *redirect_uris* if you want users to be redirected to a certain page after authenticating in an OAuth flow. You can specify multiple URLs by passing a list. Note that if you wish to use OAuth authentication with redirects, the redirect URI must be one of the URLs specified here.

Specify *to_file* to persist your app's info to a file so you can use it in the constructor. Specify *website* to give a website for your app.

Specify *session* with a requests.Session for it to be used instead of the default. This can be used to, amongst other things, adjust proxy or SSL certificate settings.

Specify *user_agent* if you want to use a specific name as *User-Agent* header, otherwise "mastodonpy" will be used.

Presently, app registration is open by default, but this is not guaranteed to be the case for all Mastodon instances in the future.

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

Mastodon.**app_verify_credentials**()
Fetch information about the current application.

Returns an *application dict*.

Added: Mastodon v2.0.0, last changed: Mastodon v2.7.2

Mastodon.**__init__**(*client_id=None*, *client_secret=None*, *access_token=None*, *api_base_url=None*, *debug_requests=False*, *ratelimit_method='wait'*, *ratelimit_pacefactor=1.1*, *request_timeout=300*, *mastodon_version=None*, *version_check_mode='created'*, *session=None*, *feature_set='mainline'*, *user_agent='mastodonpy'*, *lang=None*)
Create a new API wrapper instance based on the given *client_secret* and *client_id* on the instance given by *api_base_url*. If you give a *client_id* and it is not a file, you must also give a secret. If you specify an *access_token* then you don't need to specify a *client_id*. It is allowed to specify neither - in this case, you will be restricted to only using endpoints that do not require authentication. If a file is given as *client_id*, client ID, secret and base url are read from that file.

You can also specify an *access_token*, directly or as a file (as written by *log_in()*). If a file is given, Mastodon.py also tries to load the base URL from this file, if present. A client id and secret are not required in this case.

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 avoid hitting 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.

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*.

For fine-tuned control over the requests object use *session* with a requests.Session.

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 endpoint's behaviour has changed after the version of Mastodon that is connected has been released. If it is set to "none", version checking is disabled.

feature_set can be used to enable behaviour specific to non-mainline Mastodon API implementations. Details are documented in the functions that provide such functionality. Currently supported feature sets are *mainline*, *fedibird* and *pleroma*.

For some Mastodon instances a *User-Agent* header is needed. This can be set by parameter *user_agent*. Starting from Mastodon.py 1.5.2 *create_app()* stores the application name into the client secret file. If *client_id* points to this file, the app name will be used as *User-Agent* header as default. It is possible to modify old secret files and append a client app name to use it as a *User-Agent* name.

lang can be used to change the locale Mastodon will use to generate responses. Valid parameters are all ISO 639-1 (two letter) or for a language that has none, 639-3 (three letter) language codes. This affects some error messages (those related to validation) and trends. You can change the language using set_language().

If no other *User-Agent* is specified, "mastodonpy" will be used.

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', 'push'], to_file=None*)

Get the access token for a user.

The username is the email address 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 the OAuth flow data or the username / password credentials given are incorrect, and *MastodonAPIError* if all of the requested scopes were not granted.

For OAuth 2, obtain a code via having your user go to the URL returned by *auth_request_url()* and pass it as the code parameter. In this case, make sure to also pass the same redirect_uri parameter as you used when generating the auth request URL.

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', 'push'], force_login=False, state=None, lang=None*)

Returns the URL that a client needs to request an OAuth grant from the server.

To log in with OAuth, send your user to this URL. The user will then log in and get a code which you can pass to *log_in()*.

scopes are as in *log_in()*, redirect_uris is where the user should be redirected to after authentication. Note that *redirect_uris* must be one of the URLs given during app registration. When using urn:ietf:wg:oauth:2.0:oob, the code is simply displayed, otherwise it is added to the given URL as the "code" request parameter.

Pass force_login if you want the user to always log in even when already logged into web Mastodon (i.e. when registering multiple different accounts in an app).

state is the oauth *state* parameter to pass to the server. It is strongly suggested to use a random, nonguessable value (i.e. nothing meaningful and no incrementing ID) to preserve security guarantees. It can be left out for non-web login flows.

Pass an ISO 639-1 (two letter) or, for languages that do not have one, 639-3 (three letter) language code as *lang* to control the display language for the oauth form.

Mastodon.**set_language**(*lang*)

Set the locale Mastodon will use to generate responses. Valid parameters are all ISO 639-1 (two letter) or, for languages that do not have one, 639-3 (three letter) language codes. This affects some error messages (those related to validation) and trends.

Mastodon.**revoke_access_token**()

Revoke the oauth token the user is currently authenticated with, effectively removing the apps access and requiring the user to log in again.

Mastodon.**create_account**(*username, password, email, agreement=False, reason=None, locale='en', scopes=['read', 'write', 'follow', 'push'], to_file=None, return_detailed_error=False*)

Creates a new user account with the given username, password and email. "agreement" must be set to true (after showing the user the instance's user agreement and having them agree to it), "locale" specifies the language for the confirmation email as an ISO 639-1 (two letter) or, if a language does not have one, 639-3 (three letter) language code. *reason* can be used to specify why a user would like to join if approved-registrations mode is on.

Does not require an access token, but does require a client grant.

By default, this method is rate-limited by IP to 5 requests per 30 minutes.

Returns an access token (just like log_in), which it can also persist to to_file, and sets it internally so that the user is now logged in. Note that this token can only be used after the user has confirmed their email.

By default, the function will throw if the account could not be created. Alternately, when *return_detailed_error* is passed, Mastodon.py will return the detailed error response that the API provides (Starting from version 3.4.0 - not checked here) as an dict with error details as the second return value and the token returned as *None* in case of error. The dict will contain a text *error* values as well as a *details* value which is a dict with one optional key for each potential field (*username*, *password*, *email* and *agreement*), each if present containing a dict with an *error* category and free text *description*. Valid error categories are:

- ERR_BLOCKED - When e-mail provider is not allowed
- ERR_UNREACHABLE - When e-mail address does not resolve to any IP via DNS (MX, A, AAAA)
- ERR_TAKEN - When username or e-mail are already taken
- ERR_RESERVED - When a username is reserved, e.g. "webmaster" or "admin"
- ERR_ACCEPTED - When agreement has not been accepted
- ERR_BLANK - When a required attribute is blank
- ERR_INVALID - When an attribute is malformed, e.g. wrong characters or invalid e-mail address
- ERR_TOO_LONG - When an attribute is over the character limit
- ERR_TOO_SHORT - When an attribute is under the character requirement
- ERR_INCLUSION - When an attribute is not one of the allowed values, e.g. unsupported locale

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**email_resend_confirmation**()
Requests a re-send of the users confirmation mail for an unconfirmed logged in user.

Only available to the app that the user originally signed up with.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

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

Does not require authentication for publicly visible statuses.

Returns a *status 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 *account 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 *account 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.

This function is deprecated as of 3.0.0 and the endpoint does not exist anymore - you should just use the "card" field of the status dicts instead. Mastodon.py will try to mimic the old behaviour, but this is somewhat inefficient and not guaranteed to be the case forever.

Returns a *card dict*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

Mastodon.**status_history**(*id*)
Returns the edit history of a status as a list of status edit dicts, starting from the original form. Note that this means that a status that has been edited once will have *two* entries in this list, a status that has been edited twice will have three, and so on.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**status_source**(*id*)
Returns the source of a status for editing.

Return value is a dictionary containing exactly the parameters you could pass to *status_update()* to change nothing about the status, except *status* is *text* instead.

Mastodon.**favourites**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetch the logged-in user's favourited statuses.

This endpoint uses internal ids for pagination, passing status ids to *max_id*, *min_id*, or *since_id* will not work. Pagination functions *fetch_next()* and *fetch_previous()* should be used instead.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**bookmarks**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Get a list of statuses bookmarked by the logged-in user.

This endpoint uses internal ids for pagination, passing status ids to *max_id*, *min_id*, or *since_id* will not work. Pagination functions *fetch_next()* and *fetch_previous()* should be used instead.

Returns a list of *status dicts*.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**status_post**(*status*, *in_reply_to_id=None*, *media_ids=None*, *sensitive=False*, *visibility=None*, *spoiler_text=None*, *language=None*, *idempotency_key=None*, *content_type=None*, *scheduled_at=None*, *poll=None*, *quote_id=None*)
Post a status. Can optionally be in reply to another status and contain media.

media_ids should be a list. (If it's not, the function will turn it into one.) It can 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 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.

Specify *language* to override automatic language detection. The parameter accepts all valid ISO 639-1 (2-letter) or for languages where that do not have one, 639-3 (three letter) language codes.

You can set *idempotency_key* to a value to uniquely identify an attempt at posting a status. Even if you call this function more than once, if you call it with the same *idempotency_key*, only one status will be created.

Pass a datetime as *scheduled_at* to schedule the toot for a specific time (the time must be at least 5 minutes into the future). If this is passed, status_post returns a *scheduled status dict* instead.

Pass *poll* to attach a poll to the status. An appropriate object can be constructed using *make_poll()* . Note that as of Mastodon version 2.8.2, you can only have either media or a poll attached, not both at the same time.

Specific to "pleroma" feature set:: Specify *content_type* to set the content type of your post on Pleroma. It accepts 'text/plain' (default), 'text/markdown', 'text/html' and 'text/bbcode'. This parameter is not supported on Mastodon servers, but will be safely ignored if set.

Specific to "fedibird" feature set:: The *quote_id* parameter is a non-standard extension that specifies the id of a quoted status.

Returns a *status dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

Mastodon.**status_reply**(*to_status*, *status*, *in_reply_to_id=None*, *media_ids=None*, *sensitive=False*, *visibility=None*, *spoiler_text=None*, *language=None*, *idempotency_key=None*, *content_type=None*, *scheduled_at=None*, *poll=None*, *untag=False*)

Helper function - acts like status_post, but prepends the name of all the users that are being replied to the status text and retains CW and visibility if not explicitly overridden.

Note that *to_status* should be a *status dict* and not an ID.

Set *untag* to True if you want the reply to only go to the user you are replying to, removing every other mentioned user from the conversation.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

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

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

Usage in production code is not recommended.

Returns a *status dict* with the new status.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

Mastodon.**make_poll**(*options*, *expires_in*, *multiple=False*, *hide_totals=False*)

Generate a poll object that can be passed as the *poll* option when posting a status.

options is an array of strings with the poll options (Maximum, by default: 4), expires_in is the time in seconds for which the poll should be open. Set multiple to True to allow people to choose more than one answer. Set hide_totals to True to hide the results of the poll until it has expired.

Mastodon.**status_reblog**(*id*, *visibility=None*)
Reblog / boost a status.

The visibility parameter functions the same as in *status_post()* and allows you to reduce the visibility of a reblogged status.

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

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

Mastodon.**status_bookmark**(*id*)
Bookmark a status as the logged-in user.

Returns a *status dict* with the now bookmarked status

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**status_unbookmark**(*id*)
Unbookmark a bookmarked status for the logged-in user.

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

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

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

Returns the now-deleted status, with an added "source" attribute that contains the text that was used to compose this status (this can be used to power "delete and redraft" functionality)

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

Mastodon.**status_update**(*id*, *status=None*, *spoiler_text=None*, *sensitive=None*, *media_ids=None*, *poll=None*)

Edit a status. The meanings of the fields are largely the same as in *status_post()*, though not every field can be edited.

Note that editing a poll will reset the votes.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**scheduled_statuses**()

Fetch a list of scheduled statuses

Returns a list of *scheduled status dicts*.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**scheduled_status**(*id*)

Fetch information about the scheduled status with the given id.

Returns a *scheduled status dict*.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**scheduled_status_update**(*id*, *scheduled_at*)

Update the scheduled time of a scheduled status.

New time must be at least 5 minutes into the future.

Returns a *scheduled status dict*

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**scheduled_status_delete**(*id*)

Deletes a scheduled status.

Added: Mastodon v2.7.0, last changed: Mastodon v2.7.0

Mastodon.**media_post**(*media_file*, *mime_type=None*, *description=None*, *focus=None*, *file_name=None*, *thumbnail=None*, *thumbnail_mime_type=None*, *synchronous=False*)

Post an image, video or audio file. *media_file* can either be data or a file name. If data is passed directly, the mime type has to be specified manually, otherwise, it is determined from the file name. *focus* should be a tuple of floats between -1 and 1, giving the x and y coordinates of the images focus point for cropping (with the origin being the images center).

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

file_name can be specified to upload a file with the given name, which is ignored by Mastodon, but some other Fediverse server software will display it. If no name is specified, a random name will be generated. The filename of a file specified in media_file will be ignored.

Starting with Mastodon 3.2.0, *thumbnail* can be specified in the same way as *media_file* to upload a custom thumbnail image for audio and video files.

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

When using the v2 API (post Mastodon version 3.1.4), the *url* in the returned dict will be *null*, since attachments are processed asynchronously. You can fetch an updated dict using *media*. Pass "synchronous" to emulate the old behaviour. Not recommended, inefficient and deprecated, will eat your API quota, you know the deal.

Added: Mastodon v1.0.0, last changed: Mastodon v3.2.0

Mastodon.**media_update**(*id*, *description=None*, *focus=None*, *thumbnail=None*, *thumbnail_mime_type=None*)

Update the metadata of the media file with the given *id*. *description* and *focus* and *thumbnail* are as in *media_post()* .

Returns the updated *media dict*.

Added: Mastodon v2.3.0, last changed: Mastodon v3.2.0

Mastodon.**poll**(*id*)

Fetch information about the poll with the given id

Returns a *poll dict*.

Added: Mastodon v2.8.0, last changed: Mastodon v2.8.0

Mastodon.**account_verify_credentials**()

Fetch logged-in user's account information.

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

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

Mastodon.**me**()

Get this user's account. Synonym for *account_verify_credentials()*, does exactly the same thing, just exists because *account_verify_credentials()* has a confusing name.

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

Mastodon.**account**(*id*)

Fetch account information by user *id*.

Does not require authentication for publicly visible accounts.

Returns a *account dict*.

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

Mastodon.**account_search**(*q*, *limit=None*, *following=False*, *resolve=False*)

Fetch matching accounts. Will lookup an account remotely if the search term is in the username@domain format and not yet in the database. Set *following* to True to limit the search to users the logged-in user follows.

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.3.0

Mastodon.**account_lookup**(*acct*)

Look up an account from user@instance form (@instance allowed but not required for local accounts). Will only return accounts that the instance already knows about, and not do any webfinger requests. Use *account_search* if you need to resolve users through webfinger from remote.

Returns an *account dict*.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

Mastodon.**featured_tags**()

Return the hashtags the logged-in user has set to be featured on their profile as a list of featured tag dicts.

Returns a list of featured tag dicts.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**featured_tag_suggestions**()

Returns the logged-in user's 10 most commonly-used hashtags.

Returns a list of hashtag dicts.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**account_featured_tags**(*id*)
Get an account's featured hashtags.

Returns a list of hashtag dicts (NOT **'featured tag dicts'_**).

Added: Mastodon v3.3.0, last changed: Mastodon v3.3.0

Mastodon.**endorsements**()
Fetch list of users endorsed by the logged-in user.

Returns a list of *account dicts*.

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

Mastodon.**account_statuses**(*id*, *only_media=False*, *pinned=False*, *exclude_replies=False*, *exclude_reblogs=False*, *tagged=None*, *max_id=None*, *min_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. If *exclude_reblogs* is set, filter out all statuses that are reblogs. If *tagged* is set, return only statuses that are tagged with *tagged*. Only a single tag without a '#' is valid.

Does not require authentication for Mastodon versions after 2.7.0 (returns publicly visible statuses in that case), for publicly visible accounts.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.8.0

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**account_familiar_followers**(*id*)
Find followers for the account given by id (can be a list) that also follow the logged in account.

Returns a list of familiar follower dicts

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**account_lists**(*id*)
Get all of the logged-in user's lists which the specified user is a member of.

Returns a list of list dicts.

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

Mastodon.**account_update_credentials**(*display_name=None*, *note=None*, *avatar=None*, *avatar_mime_type=None*, *header=None*, *header_mime_type=None*, *locked=None*, *bot=None*, *discoverable=None*, *fields=None*)
Update the profile for the currently logged-in user.

note is the user's bio.

avatar and 'header' are images. As with media uploads, it is possible to either pass image data and a mime type, or a filename of an image file, for either.

locked specifies whether the user needs to manually approve follow requests.

bot specifies whether the user should be set to a bot.

discoverable specifies whether the user should appear in the user directory.

fields can be a list of up to four name-value pairs (specified as tuples) to appear as semi-structured information in the user's profile.

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

Added: Mastodon v1.1.1, last changed: Mastodon v3.1.0

Mastodon.**account_pin**(*id*)
Pin / endorse a user.

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

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

Mastodon.**account_unpin**(*id*)
Unpin / un-endorse a user.

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

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

Mastodon.**account_note_set**(*id*, *comment*)
Set a note (visible to the logged in user only) for the given account.

Returns a *status dict* with the *note* updated.

Added: Mastodon v3.2.0, last changed: Mastodon v3.2.0

Mastodon.**featured_tag_create**(*name*)
Creates a new featured hashtag displayed on the logged-in user's profile.

Returns a *featured tag dict* with the newly featured tag.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**featured_tag_delete**(*id*)
Deletes one of the logged-in user's featured hashtags.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**status_pin**(*id*)
Pin a status for the logged-in user.

Returns a *status dict* with the now pinned status

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

Mastodon.**status_unpin**(*id*)
Unpin a pinned status for the logged-in user.

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

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

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

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.**follows**(*uri*)
Follow a remote user with username given in username@domain form.

Returns a *account dict*.

Deprecated - avoid using this. Currently uses a backwards compat implementation that may or may not work properly.

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

Mastodon.**follow_requests**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetch the logged-in user's incoming follow requests.

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**suggestions**()
Fetch follow suggestions for the logged-in user.

Returns a list of *account dicts*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**account_follow**(*id*, *reblogs=True*, *notify=False*)
Follow a user.

Set *reblogs* to False to hide boosts by the followed user. Set *notify* to True to get a notification every time the followed user posts.

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

Added: Mastodon v1.0.0, last changed: Mastodon v3.3.0

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.**follow_request_authorize**(*id*)
Accept an incoming follow request.

Returns the updated *relationship dict* for the requesting account.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

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

Returns the updated *relationship dict* for the requesting account.

Added: Mastodon v1.0.0, last changed: Mastodon v3.0.0

Mastodon.**suggestion_delete**(*account_id*)
Remove the user with the given *account_id* from the follow suggestions.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

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

Returns a list of *account dicts*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.6.0

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

Returns a list of *account dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v2.6.0

Mastodon.**domain_blocks**(*max_id=None*, *min_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 v2.6.0

Mastodon.**account_mute**(*id*, *notifications=True*, *duration=None*)
Mute a user.

Set *notifications* to False to receive notifications even though the user is muted from timelines. Pass a *duration* in seconds to have Mastodon automatically lift the mute after that many seconds.

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

Added: Mastodon v1.1.0, last changed: Mastodon v2.4.3

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_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_remove_from_followers**(*id*)
Remove a user from the logged in users followers (i.e. make them unfollow the logged in user / "softblock" them).

Returns a *relationship dict* reflecting the updated following status.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

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

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*, *min_id=None*, *since_id=None*, *limit=None*)
Get the accounts that are on the given list.

Returns a list of *account dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v2.6.0

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

Mastodon.**timeline**(*timeline='home'*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
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.

The default timeline is the "home" timeline.

Specify *only_media* to only get posts with attached media. Specify *local* to only get local statuses, and *remote* to only get remote statuses. Some options are mutually incompatible as dictated by logic.

May or may not require authentication depending on server settings and what is specifically requested.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_home**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
Convenience method: Fetches the logged-in user's home timeline (i.e. followed users and self). Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_local**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*)
Convenience method: Fetches the local / instance-wide timeline, not including replies. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_public**(*max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
Convenience method: Fetches the public / visible-network / federated timeline, not including replies. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_hashtag**(*hashtag*, *local=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *remote=False*)
Convenience method: Fetch a timeline of toots with a given hashtag. The hashtag parameter should not contain the leading #. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v1.0.0, last changed: Mastodon v3.1.4

Mastodon.**timeline_list**(*id*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *only_media=False*, *local=False*, *remote=False*)
Convenience method: Fetches a timeline containing all the toots by users in a given list. Params as in *timeline()*.

Returns a list of *status dicts*.

Added: Mastodon v2.1.0, last changed: Mastodon v3.1.4

Mastodon.**instance**()
Retrieve basic information about the instance, including the URI and administrative contact email.

Does not require authentication unless locked down by the administrator.

Returns an *instance dict*.

Added: Mastodon v1.1.0, last changed: Mastodon v2.3.0

Mastodon.**instance_activity**()
Retrieve activity stats about the instance. May be disabled by the instance administrator - throws a MastodonNotFoundError in that case.

Activity is returned for 12 weeks going back from the current week.

Returns a list of activity dicts.

Added: Mastodon v2.1.2, last changed: Mastodon v2.1.2

Mastodon.**instance_peers**()
Retrieve the instances that this instance knows about. May be disabled by the instance administrator - throws a MastodonNotFoundError in that case.

Returns a list of URL strings.

Added: Mastodon v2.1.2, last changed: Mastodon v2.1.2

Mastodon.**instance_health**()
Basic health check. Returns True if healthy, False if not.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**instance_nodeinfo**(*schema='http://nodeinfo.diaspora.software/ns/schema/2.0'*)
Retrieves the instance's nodeinfo information.

For information on what the nodeinfo can contain, see the nodeinfo specification: https://github.com/jhass/nodeinfo . By default, Mastodon.py will try to retrieve the version 2.0 schema nodeinfo.

To override the schema, specify the desired schema with the *schema* parameter.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**instance_rules**()
Retrieve instance rules.

Returns a list of *id* + *text* dicts, same as the *rules* field in the instance dicts.

Added: Mastodon v3.4.0, last changed: Mastodon v3.4.0

Mastodon.**directory**(*offset=None*, *limit=None*, *order=None*, *local=None*)
Fetch the contents of the profile directory, if enabled on the server.

offset how many accounts to skip before returning results. Default 0.

limit how many accounts to load. Default 40.

order **"active" to sort by most recently posted statuses (default) or** "new" to sort by most recently created profiles.

local True to return only local accounts.

Returns a list of *account dicts*.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**custom_emojis**()
Fetch the list of custom emoji the instance has installed.

Does not require authentication unless locked down by the administrator.

Returns a list of emoji dicts.

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

Mastodon.**announcements**()
Fetch currently active announcements.

Returns a list of announcement dicts.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**announcement_dismiss**(*id*)
Set the given annoucement to read.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**announcement_reaction_create**(*id*, *reaction*)
Add a reaction to an announcement. *reaction* can either be a unicode emoji or the name of one of the instances custom emoji.

Will throw an API error if the reaction name is not one of the allowed things or when trying to add a reaction that the user has already added (adding a reaction that a different user added is legal and increments the count).

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**announcement_reaction_delete**(*id*, *reaction*)
Remove a reaction to an announcement.

Will throw an API error if the reaction does not exist.

Added: Mastodon v3.1.0, last changed: Mastodon v3.1.0

Mastodon.**trending_tags**(*limit=None*, *lang=None*)
Fetch trending-hashtag information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Does not require authentication unless locked down by the administrator.

Important versioning note: This endpoint does not exist for Mastodon versions between 2.8.0 (inclusive) and 3.0.0 (exclusive).

Pass *lang* to override the global locale parameter, which may affect trend ordering.

Returns a list of hashtag dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**trending_statuses**(*limit=None*, *lang=None*)
Fetch trending-status information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Pass *lang* to override the global locale parameter, which may affect trend ordering.

Returns a list of *status dicts*, sorted by the instances's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**trending_links**(*limit=None*, *lang=None*)
Fetch trending-link information, if the instance provides such information.

Specify *limit* to limit how many results are returned (the maximum number of results is 10, the endpoint is not paginated).

Returns a list of card dicts, sorted by the instances's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**trends**(*limit=None*)
Old alias for *trending_tags()*

Deprecated. Please use *trending_tags()* instead.

Added: Mastodon v2.4.3, last changed: Mastodon v3.5.0

Mastodon.**search**(*q*, *resolve=True*, *result_type=None*, *account_id=None*, *offset=None*, *min_id=None*, *max_id=None*, *exclude_unreviewed=True*)
Fetch matching hashtags, accounts and statuses. Will perform webfinger lookups if resolve is True. Full-text search is only enabled if the instance supports it, and is restricted to statuses the logged-in user wrote or was mentioned in.

result_type can be one of "accounts", "hashtags" or "statuses", to only search for that type of object.

Specify *account_id* to only get results from the account with that id.

offset, *min_id* and *max_id* can be used to paginate.

exclude_unreviewed can be used to restrict search results for hashtags to only those that have been reviewed by moderators. It is on by default. When using the v1 search API (pre 2.4.1), it is ignored.

Will use search_v1 (no tag dicts in return values) on Mastodon versions before 2.4.1), search_v2 otherwise. Parameters other than resolve are only available on Mastodon 2.8.0 or above - this function will throw a MastodonVersionError if you try to use them on versions before that. Note that the cached version number will be used for this to avoid uneccesary requests.

Returns a *search result dict*, with tags as **'hashtag dicts'_**.

Added: Mastodon v1.1.0, last changed: Mastodon v2.8.0

`Mastodon.`**`search_v2`**(*q*, *resolve=True*, *result_type=None*, *account_id=None*, *offset=None*, *min_id=None*, *max_id=None*, *exclude_unreviewed=True*)

Identical to *search_v1()*, except in that it returns tags as hashtag dicts, has more parameters, and resolves by default.

For more details documentation, please see *search()*

Returns a *search result dict*.

Added: Mastodon v2.4.1, last changed: Mastodon v2.8.0

`Mastodon.`**`notifications`**(*id=None*, *account_id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*, *exclude_types=None*, *types=None*, *mentions_only=None*)

Fetch notifications (mentions, favourites, reblogs, follows) for the logged-in user. Pass *account_id* to get only notifications originating from the given account.

There are different types of notifications:

- *follow* - A user followed the logged in user
- *follow_request* - A user has requested to follow the logged in user (for locked accounts)
- *favourite* - A user favourited a post by the logged in user
- *reblog* - A user reblogged a post by the logged in user
- *mention* - A user mentioned the logged in user
- *poll* - A poll the logged in user created or voted in has ended
- *update* - A status the logged in user has reblogged (and only those, as of 4.0.0) has been edited
- *status* - A user that the logged in user has enabned notifications for has enabled *notify* (see *account_follow()*)
- *admin.sign_up* - For accounts with appropriate permissions (TODO: document which those are when adding the permission API): A new user has signed up
- *admin.report* - For accounts with appropriate permissions (TODO: document which those are when adding the permission API): A new report has been received

Parameters *exclude_types* and *types* are array of these types, specifying them will in- or exclude the types of notifications given. It is legal to give both parameters at the same tine, the result will then be the intersection of the results of both filters. Specifying *mentions_only* is a deprecated way to set *exclude_types* to all but mentions.

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

Returns a list of notification dicts.

Added: Mastodon v1.0.0, last changed: Mastodon v3.5.0

Mastodon.**notifications_clear**()
Clear out a user's 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 v2.9.2

Mastodon.**conversations_read**(*id*)
Marks a single conversation as read.

Returns the updated *conversation dict*.

Added: Mastodon v2.6.0, last changed: Mastodon v2.6.0

Mastodon.**filters**()
Fetch all of the logged-in user's filters.

Returns a list of filter dicts. Not paginated.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter**(*id*)
Fetches information about the filter with the specified *id*.

Returns a *filter dict*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filters_apply**(*objects*, *filters*, *context*)
Helper function: Applies a list of filters to a list of either statuses or notifications and returns only those matched by none. This function will apply all filters that match the context provided in *context*, i.e. if you want to apply only notification-relevant filters, specify 'notifications'. Valid contexts are 'home', 'notifications', 'public' and 'thread'.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter_create**(*phrase*, *context*, *irreversible=False*, *whole_word=True*, *expires_in=None*)
Creates a new keyword filter. *phrase* is the phrase that should be filtered out, *context* specifies from where to filter the keywords. Valid contexts are 'home', 'notifications', 'public' and 'thread'.

Set *irreversible* to True if you want the filter to just delete statuses server side. This works only for the 'home' and 'notifications' contexts.

Set *whole_word* to False if you want to allow filter matches to start or end within a word, not only at word boundaries.

Set *expires_in* to specify for how many seconds the filter should be kept around.

Returns the *filter dict* of the newly created filter.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter_update**(*id*, *phrase=None*, *context=None*, *irreversible=None*, *whole_word=None*, *expires_in=None*)
Updates the filter with the given *id*. Parameters are the same as in *filter_create()*.

Returns the *filter dict* of the updated filter.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**filter_delete**(*id*)
Deletes the filter with the given *id*.

Added: Mastodon v2.4.3, last changed: Mastodon v2.4.3

Mastodon.**push_subscription**()
Fetch the current push subscription the logged-in user has for this app.

Returns a *push subscription dict*.

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

Mastodon.**push_subscription_set**(*endpoint*, *encrypt_params*, *follow_events=None*, *favourite_events=None*, *reblog_events=None*, *mention_events=None*, *poll_events=None*, *follow_request_events=None*, *status_events=None*, *policy='all'*)
Sets up or modifies the push subscription the logged-in user has for this app.

endpoint is the endpoint URL mastodon should call for pushes. Note that mastodon requires https for this URL. *encrypt_params* is a dict with key parameters that allow the server to encrypt data for you: A public key *pubkey* and a shared secret *auth*. You can generate this as well as the corresponding private key using the *push_subscription_generate_keys()* function.

policy controls what sources will generate webpush events. Valid values are *all*, *none*, *follower* and *followed*.

The rest of the parameters controls what kind of events you wish to subscribe to.

Returns a *push subscription dict*.

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

Mastodon.**push_subscription_update**(*follow_events=None*, *favourite_events=None*, *reblog_events=None*, *mention_events=None*, *poll_events=None*, *follow_request_events=None*)
Modifies what kind of events the app wishes to subscribe to.

Returns the updated *push subscription dict*.

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

Mastodon.**push_subscription_generate_keys**()
Generates a private key, public key and shared secret for use in webpush subscriptions.

Returns two dicts: One with the private key and shared secret and another with the public key and shared secret.

Mastodon.**push_subscription_decrypt_push**(*data*, *decrypt_params*, *encryption_header*, *crypto_key_header*)
Decrypts *data* received in a webpush request. Requires the private key dict from *push_subscription_generate_keys()* (*decrypt_params*) as well as the Encryption and server Crypto-Key headers from the received webpush

Returns the decoded webpush as a *push notification dict*.

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

Mastodon.**stream_user**(*listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)
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*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*, *local=False*, *remote=False*)
Streams public events.

Set *local* to True to only get local statuses. Set *remote* to True to only get remote statuses.

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

Mastodon.**stream_local**(*listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)
Streams local public events.

This function is deprecated. Please use stream_public() with parameter *local* set to True instead.

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

Mastodon.**stream_hashtag**(*tag*, *listener*, *local=False*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)
Stream for all public statuses for the hashtag 'tag' seen by the connected instance.

Set *local* to True to only get local statuses.

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

Mastodon.**stream_list**(*id*, *listener*, *run_async=False*, *timeout=300*, *reconnect_async=False*, *reconnect_async_wait_sec=5*)
Stream events for the current user, restricted to accounts on the given list.

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

Mastodon.**stream_healthy**()
Returns without True if streaming API is okay, False or raises an error otherwise.

Added: Mastodon v2.5.0, last changed: Mastodon v2.5.0

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 *status dict* describing the status.

StreamListener.**on_notification**(*notification*)
A new notification. *notification* is the parsed *notification dict* describing the notification.

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

StreamListener.**on_conversation**(*conversation*)
A direct message (in the direct stream) has been received. *conversation* is the parsed *conversation dict* dictionary describing the conversation

StreamListener.**on_status_update**(*status*)
A status has been edited. 'status' is the parsed JSON dictionary describing the updated status.

StreamListener.**on_unknown_event**(*name*, *unknown_event=None*)
An unknown mastodon API event has been received. The name contains the event-name and unknown_event contains the content of the unknown event.

StreamListener.**on_abort**(*err*)
There was a connection error, read timeout or other error fatal to the streaming connection. The exception object about to be raised is passed to this function for reference.

Note that the exception will be raised properly once you return from this function, so if you are using this handler to reconnect, either never return or start a thread and then catch and ignore the exception.

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.

Mastodon.**markers_get**(*timeline=['home']*)
Get the last-read-location markers for the specified timelines. Valid timelines are the same as in *timeline()*

Note that despite the singular name, *timeline* can be a list.

Returns a dict of read marker dicts, keyed by timeline name.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**markers_set**(*timelines*, *last_read_ids*)
Set the "last read" marker(s) for the given timeline(s) to the given id(s)

Note that if you give an invalid timeline name, this will silently do nothing.

Returns a dict with the updated read marker dicts, keyed by timeline name.

Added: Mastodon v3.0.0, last changed: Mastodon v3.0.0

Mastodon.**reports**()
Fetch a list of reports made by the logged-in user.

Returns a list of report dicts.

Warning: This method has now finally been removed, and will not work on Mastodon versions 2.5.0 and above.

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

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.

Mastodon.**decode_blurhash**(*media_dict*, *out_size=(16, 16)*, *size_per_component=True*, *return_linear=True*)
Basic media-dict blurhash decoding.

out_size is the desired result size in pixels, either absolute or per blurhash component (this is the default).

By default, this function will return the image as linear RGB, ready for further scaling operations. If you want to display the image directly, set return_linear to False.

Returns the decoded blurhash image as a three-dimensional list: [height][width][3], with the last dimension being RGB colours.

For further info and tips for advanced usage, refer to the documentation for the blurhash module: https://github.com/halcy/blurhash-python

Mastodon.**admin_accounts_v2**(*origin=None*, *by_domain=None*, *status=None*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *permissions=None*, *invited_by=None*, *role_ids=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)
Fetches a list of accounts that match given criteria. By default, local accounts are returned.

- Set *origin* to "local" or "remote" to get only local or remote accounts.
- Set *by_domain* to a domain to get only accounts from that domain.
- Set *status* to one of "active", "pending", "disabled", "silenced" or "suspended" to get only accounts with that moderation status (default: active)
- Set *username* to a string to get only accounts whose username contains this string.
- Set *display_name* to a string to get only accounts whose display name contains this string.
- Set *email* to an email to get only accounts with that email (this only works on local accounts).
- Set *ip* to an ip (as a string, standard v4/v6 notation) to get only accounts whose last active ip is that ip (this only works on local accounts).
- Set *permissions* to "staff" to only get accounts with staff permissions.
- Set *invited_by* to an account id to get only accounts invited by this user.
- Set *role_ids* to a list of role IDs to get only accounts with those roles.

Returns a list of admin account dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v4.0.0

Mastodon.**admin_accounts**(*remote=False*, *by_domain=None*, *status='active'*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *staff_only=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Currently a synonym for admin_accounts_v1, now deprecated. You are strongly encouraged to use admin_accounts_v2 instead, since this one is kind of bad.

!!!!! This function may be switched to calling the v2 API in the future. This is your warning. If you want to keep using v1, use it explicitly. !!!!!

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_accounts_v1**(*remote=False*, *by_domain=None*, *status='active'*, *username=None*, *display_name=None*, *email=None*, *ip=None*, *staff_only=False*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches a list of accounts that match given criteria. By default, local accounts are returned.

- Set *remote* to True to get remote accounts, otherwise local accounts are returned (default: local accounts)
- Set *by_domain* to a domain to get only accounts from that domain.
- Set *status* to one of "active", "pending", "disabled", "silenced" or "suspended" to get only accounts with that moderation status (default: active)
- Set *username* to a string to get only accounts whose username contains this string.
- Set *display_name* to a string to get only accounts whose display name contains this string.
- Set *email* to an email to get only accounts with that email (this only works on local accounts).
- Set *ip* to an ip (as a string, standard v4/v6 notation) to get only accounts whose last active ip is that ip (this only works on local accounts).
- Set *staff_only* to True to only get staff accounts (this only works on local accounts).

Note that setting the boolean parameters to False does not mean "give me users to which this does not apply" but instead means "I do not care if users have this attribute".

Deprecated in Mastodon version 3.5.0.

Returns a list of admin account dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account**(*id*)
Fetches a single *admin account dict* for the user with the given id.

Returns that dict.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_enable**(*id*)
Reenables login for a local account for which login has been disabled.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_approve**(*id*)
Approves a pending account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_reject**(*id*)
Rejects and deletes a pending account.

Returns the updated *admin account dict* for the account that is now gone.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_unsilence**(*id*)
Unsilences an account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_unsuspend**(*id*)
Unsuspends an account.

Returns the updated *admin account dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_account_moderate**(*id*, *action=None*, *report_id=None*, *warning_preset_id=None*, *text=None*, *send_email_notification=True*)
Perform a moderation action on an account.

Valid actions are:

- "disable" - for a local user, disable login.
- "silence" - hide the users posts from all public timelines.
- "suspend" - irreversibly delete all the user's posts, past and future.
- "sensitive" - forcce an accounts media visibility to always be sensitive.

If no action is specified, the user is only issued a warning.

Specify the id of a report as *report_id* to close the report with this moderation action as the resolution. Specify *warning_preset_id* to use a warning preset as the notification text to the user, or *text* to specify text directly. If both are specified, they are concatenated (preset first). Note that there is currently no API to retrieve or create warning presets.

Set *send_email_notification* to False to not send the user an email notification informing them of the moderation action.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_reports**(*resolved=False*, *account_id=None*, *target_account_id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches the list of reports.

Set *resolved* to True to search for resolved reports. *account_id* and *target_account_id* can be used to get reports filed by or about a specific user.

Returns a list of report dicts.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report**(*id*)

Fetches the report with the given id.

Returns a *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report_assign**(*id*)

Assigns the given report to the logged-in user.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report_unassign**(*id*)

Unassigns the given report from the logged-in user.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report_reopen**(*id*)

Reopens a closed report.

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_report_resolve**(*id*)

Marks a report as resolved (without taking any action).

Returns the updated *report dict*.

Added: Mastodon v2.9.1, last changed: Mastodon v2.9.1

Mastodon.**admin_trending_tags**(*limit=None*)

Admin version of *trending_tags()*. Includes unapproved tags.

Returns a list of hashtag dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_trending_statuses**()

Admin version of *trending_statuses()*. Includes unapproved tags.

Returns a list of *status dicts*, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_trending_links**()

Admin version of *trending_links()*. Includes unapproved tags.

Returns a list of card dicts, sorted by the instance's trending algorithm, descending.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_domain_blocks**(*id=None*, *max_id=None*, *min_id=None*, *since_id=None*, *limit=None*)

Fetches a list of blocked domains. Requires scope *admin:read:domain_blocks*.

Provide an *id* to fetch a specific domain block based on its database id.

Returns a list of admin domain block dicts, raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_create_domain_block**(*domain: str*, *severity: str = None*, *reject_media: bool = None*, *reject_reports: bool = None*, *private_comment: str = None*, *public_comment: str = None*, *obfuscate: bool = None*)

Perform a moderation action on a domain. Requires scope *admin:write:domain_blocks*.

Valid severities are:

- "silence" - hide all posts from federated timelines and do not show notifications to local users from the remote instance's users unless they are following the remote user.
- "suspend" - deny interactions with this instance going forward. This action is reversible.
- "limit" - generally used with reject_media=true to force reject media from an instance without silencing or suspending..

If no action is specified, the domain is only silenced. *domain* is the domain to block. Note that using the top level domain will also imapct all subdomains. ie, example.com will also impact subdomain.example.com. *reject_media* will not download remote media on to your local instance media storage. *reject_reports* ignores all reports from the remote instance. *private_comment* sets a private admin comment for the domain. *public_comment* sets a publicly available comment for this domain, which will be available to local users and may be available to everyone depending on your settings. *obfuscate* censors some part of the domain name. Useful if the domain name contains unwanted words like slurs.

Returns the new domain block as an *admin domain block dict*.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_update_domain_block**(*id*, *severity: str = None*, *reject_media: bool = None*, *reject_reports: bool = None*, *private_comment: str = None*, *public_comment: str = None*, *obfuscate: bool = None*)

Modify existing moderation action on a domain. Requires scope *admin:write:domain_blocks*.

Valid severities are:

- "silence" - hide all posts from federated timelines and do not show notifications to local users from the remote instance's users unless they are following the remote user.
- "suspend" - deny interactions with this instance going forward. This action is reversible.
- "limit" - generally used with reject_media=true to force reject media from an instance without silencing or suspending.

If no action is specified, the domain is only silenced. *domain* is the domain to block. Note that using the top level domain will also imapct all subdomains. ie, example.com will also impact subdomain.example.com. *reject_media* will not download remote media on to your local instance media storage. *reject_reports* ignores all reports from the remote instance. *private_comment* sets a private admin comment for the domain. *public_comment* sets a publicly available comment for this domain, which will be available to local users and may be available to everyone depending on your settings. *obfuscate* censors some part of the domain name. Useful if the domain name contains unwanted words like slurs.

Returns the modified domain block as an *admin domain block dict*, raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_delete_domain_block**(*id=None*)

Removes moderation action against a given domain. Requires scope *admin:write:domain_blocks*.

Provide an *id* to remove a specific domain block based on its database id.

Raises a *MastodonAPIError* if the specified block does not exist.

Added: Mastodon v4.0.0, last changed: Mastodon v4.0.0

Mastodon.**admin_measures**(*start_at*, *end_at*, *active_users=False*, *new_users=False*, *interactions=False*, *opened_reports=False*, *resolved_reports=False*, *tag_accounts=None*, *tag_uses=None*, *tag_servers=None*, *instance_accounts=None*, *instance_media_attachments=None*, *instance_reports=None*, *instance_statuses=None*, *instance_follows=None*, *instance_followers=None*)

Retrieves numerical instance information for the time period (at day granularity) between *start_at* and *end_at*.

- *active_users*: Pass true to retrieve the number of active users on your instance within the time period
- *new_users*: Pass true to retrieve the number of users who joined your instance within the time period
- *interactions*: Pass true to retrieve the number of interactions (favourites, boosts, replies) on local statuses within the time period
- *opened_reports*: Pass true to retrieve the number of reports filed within the time period
- *resolved_reports* = Pass true to retrieve the number of reports resolved within the time period
- *tag_accounts*: Pass a tag ID to get the number of accounts which used that tag in at least one status within the time period
- *tag_uses*: Pass a tag ID to get the number of statuses which used that tag within the time period
- *tag_servers*: Pass a tag ID to to get the number of remote origin servers for statuses which used that tag within the time period
- *instance_accounts*: Pass a domain to get the number of accounts originating from that remote domain within the time period
- *instance_media_attachments*: Pass a domain to get the amount of space used by media attachments from that remote domain within the time period
- *instance_reports*: Pass a domain to get the number of reports filed against accounts from that remote domain within the time period
- *instance_statuses*: Pass a domain to get the number of statuses originating from that remote domain within the time period
- *instance_follows*: Pass a domain to get the number of accounts from a remote domain followed by that local user within the time period
- *instance_followers*: Pass a domain to get the number of local accounts followed by accounts from that remote domain within the time period

This API call is relatively expensive - watch your servers load if you want to get a lot of statistical data. Especially the instance_statuses stats might take a long time to compute and, in fact, time out.

There is currently no way to get tag IDs implemented in Mastodon.py, because the Mastodon public API does not implement one. This will be fixed in a future release.

Returns a list of admin measure dicts.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_dimensions**(*start_at*, *end_at*, *limit=None*, *languages=False*, *sources=False*, *servers=False*, *space_usage=False*, *software_versions=False*, *tag_servers=None*, *tag_languages=None*, *instance_accounts=None*, *instance_languages=None*)

Retrieves primarily categorical instance information for the time period (at day granularity) between *start_at* and *end_at*.

- *languages*: Pass true to get the most-used languages on this server
- *sources*: Pass true to get the most-used client apps on this server
- *servers*: Pass true to get the remote servers with the most statuses
- *space_usage*: Pass true to get the how much space is used by different components your software stack
- *software_versions*: Pass true to get the version numbers for your software stack
- *tag_servers*: Pass a tag ID to get the most-common servers for statuses including a trending tag
- *tag_languages*: Pass a tag ID to get the most-used languages for statuses including a trending tag
- *instance_accounts*: Pass a domain to get the most-followed accounts from a remote server
- *instance_languages*: Pass a domain to get the most-used languages from a remote server

Pass *limit* to set how many results you want on queries where that makes sense.

This API call is relatively expensive - watch your servers load if you want to get a lot of statistical data.

There is currently no way to get tag IDs implemented in Mastodon.py, because the Mastodon public API does not implement one. This will be fixed in a future release.

Returns a list of admin dimension dicts.

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Mastodon.**admin_retention**(*start_at*, *end_at*, *frequency='day'*)

Gets user retention statistics (at *frequency* - "day" or "month" - granularity) between *start_at* and *end_at*.

Returns a list of admin retention dicts

Added: Mastodon v3.5.0, last changed: Mastodon v3.5.0

Python Module Index

m

Index

Symbols

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

V