# aPRAW

***Release 0.6.9-alpha***

**RaviAnand Mohabir**

**Aug 31, 2021**

# GETTING STARTED

aPRAW is an asynchronous Reddit API wrapper.

**Features:**

- Modern Pythonic design using *async*/*await* syntax.
- Automatic handling of rate-limits.
- Built-in listings and streams for list endpoints.
- Compatible with previous *praw.ini* files.
- Proper OAuth2 support.

# CHAPTER ONE

# COMMUNITY AND SUPPORT

If you have any questions regarding aPRAW and its usage…

- Join the /r/aPRAW subreddit
  - Feel free to post a question in the questions thread or make your own post if it could start a big discussion!
- Join the aPRAW Discord server
  - Use the `#general` chat for discussion about the library and talking to other users.
  - Use the `#questions` to post questions. The developers will try to get back to you as quickly as possible, but other users can help as well!
  - Use the `#ideas` if you have any ideas for the framework but don't know how to implement them, or just want to throw in the suggestion.

CHAPTER

TWO

# DOCUMENTATION CONTENTS

This is the documentation for aPRAW, a wrapper library for Python to aid in performing asynchronous requests to the Reddit API and interacting with its data. It's split into the following sections.

## 2.1 Introduction

aPRAW serves as an asynchronous alternative to PRAW, and offers certain features and a more modern class design. Those familiar with PRAW will be able to use many features without much additional changes to the code, besides the usage of `async` and `await` syntax.

aPRAW was specifically built with Discord bots in mind, so those interested in creating a Discord bot with Discord.py and combining Reddit streams should be able to make use of its asynchronous functionalities.

### 2.1.1 Prerequisites

aPRAW works with Python 3.6 or higher.

### 2.1.2 Installing

aPRAW can be installed directly from PyPi:

```
$ pip install aPRAW
```

## 2.2 Quickstart

This section contains a small guide to get started with using aPRAW and its various features.

**Contents**

### 2.2.1 Creating a Reddit Instance

Currently aPRAW only supports the use of a script auth flow to log in to Reddit and perform requests. Read-only modes as well as the application flow are WIP.

To obtain a `client_id` and `client_secret` for your application, head to Reddit's App Preferences and create a new app. Follow the guidelines on Reddit's Quick Start Example to obtain your credentials.

Those credentials can now be used to create a Reddit instance:

```python
import apraw


# instantiate a `Reddit` instance
# you can also supply a key to an entry within a praw.ini
# file, making your login compatible with praw as well
reddit = apraw.Reddit(username="USERNAME", password="PASSWORD",
                      client_id="CLIENT_ID", client_secret="CLIENT_SECRET",
                      user_agent="USERAGENT")
```

Those previously making use of a `praw.ini` file can continue to do so, by specifying the key that was used for the client in place of the credentials. aPRAW will then automatically search for the file and save those credentials.

For more information on `praw.ini` files visit PRAW's documentation.

### 2.2.2 Running Asynchronous Code

Since most of aPRAW's code are asynchronous functions or generators, you will want to add your tasks to an event loop such as the `asyncio` one.

For that do the following:

```python
import apraw
import asyncio

# instantiate a `Reddit` instance
reddit = apraw.Reddit(client_id="CLIENT_ID", client_secret="CLIENT_SECRET",
                    password="PASSWORD", user_agent="USERAGENT",
                    username="USERNAME")

async def scan_posts():
  # get an instance of a subreddit
  subreddit = await reddit.subreddit("aprawtest")

  # loop through new posts
  async for submission in subreddit.new():
    print(submission.title)

if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    # get the asyncio event loop
    loop = asyncio.get_event_loop()

    # add scan_posts() to the queue and run it
    loop.run_until_complete(scan_posts())
```

### 2.2.3 Basic Concepts

aPRAW assumes that all the Reddit items know the logged-in Reddit instance. When grabbing items by using the built-in functions, this will be done automatically through dependency injection.

#### Instantiating Models

Most items can be retrieved from the base Reddit object like so:

```
# instantiate a `Reddit` instance
reddit = apraw.Reddit(client_id="CLIENT_ID", client_secret="CLIENT_SECRET",
                    password="PASSWORD", user_agent="USERAGENT",
                    username="USERNAME")

# grab an instance of the /r/aPRAWTest subreddit
subreddit = await reddit.subreddit("aprawtest")

# grab an instance of the /u/aPRAWBot Redditor
redditor = await reddit.redditor("aprawbot")

# grab a test submission made on /r/aPRAWTest
submission = await reddit.submission("h7mna9")

# grab a test comment made on /r/aPRAWTest
comment = await reddit.comment("fulsybg")
```

#### Looping Through Items

Most endpoints returning list or "*listings*" of items are represented by async generators in aPRAW. To grab a set of new posts on a subreddit try this:

```
# get an instance of a subreddit
subreddit = await reddit.subreddit("aprawtest")

# loop through new posts
async for submission in subreddit.new():
    print(submission.id)
```

In cases where *`ListingGenerator`* is used, `**kwargs` can be passed into the endpoint as well.

#### Streaming Items

*ListingGenerator* has a built-in `stream()` method that will poll the Reddit API endpoint it's mapped to, and yield items as they come. This is done in a very efficient manner with an internal tracker for items, an exponential function to increase wait times and the use of `asyncio.sleep()` to ensure non-blocking streams.

Polling an endpoint with *ListingGenerator* is as simple as writing:

```
# get an instance of a subreddit
subreddit = await reddit.subreddit("aprawtest")

# stream new posts
async for submission in subreddit.new.stream():
    print(submission.id)
```

## 2.3 Reddit

### 2.3.1 User

This section describes `User` class as well as `AuthenticatedUser` that contain information about the logged-in user and request credentials.

**Contents**

**class** `apraw.models.`**`User`**(*reddit: Reddit*, *username: str*, *password: str*, *client_id: str*, *client_secret: str*, *user_agent: str*)

A class to store the authentication credentials and handle ratelimit information.

**reddit: Reddit** The *Reddit* instance with which requests are made.

**username: str** The username given to the Reddit instance or obtained via `praw.ini`.

**password: str** The password given to the Reddit instance or obtained via `praw.ini`.

**client_id: str** The client ID given to the Reddit instance or obtained via `praw.ini`.

**client_secret: str** The client secret given to the Reddit instance or obtained via `praw.ini`.

**user_agent: str** The user agent given to the Reddit instance or defaulted to aPRAW's version.

**password_grant: str** The data to be used when making a token request with the 'password' `grant_type`.

**access_data: Dict** A dictionary containing the access token and user agent for request headers.

**token_expires: datetime** The datetime on which the previously retrieved token will expire. Defaults to the past to obtain a token immediately the first time.

**ratelimit_remaining: int** The number of requests remaining in the current ratelimit window.

**ratelimit_used: int** The number of requests previously used in the current ratelimit window.

**ratelimit_reset: datetime** The datetime on which the ratelimit window will be reset.

**async auth_session()** → aiohttp.client.ClientSession

Retrieve an `aiohttp.ClientSesssion` with which the authentication token can be obtained.

**Returns** **session** – The session using the BasicAuth setup to obtain tokens with.

**Return type** aiohttp.ClientSession

**async client_session()** → aiohttp.client.ClientSession

Retrieve the `aiohttp.ClientSesssion` with which regular requests are made.

**Returns** **session** – The session with which requests should be made.

**Return type** aiohttp.ClientSession

**async me()** → *apraw.models.user.AuthenticatedUser*

Retrieve an instance of `AuthenticatedUser` for the logged-in user.

**Returns** **user** – The logged-in user.

**Return type** *AuthenticatedUser*

### AuthenticatedUser

**class apraw.models.AuthenticatedUser**(*reddit: Reddit*, *data: Dict*)

The model representing the logged-in user.

This model inherits from `Redditor` and thus all its attributes and features. View those docs for further information.

**reddit: Reddit** The `Reddit` instance with which requests are made.

**data: Dict** The data obtained from the /about endpoint.

**async karma()** → List[*apraw.models.user.Karma*]

Retrieve the karma breakdown for the logged-in user.

**Returns** **karma** – The parsed `KarmaList` for the logged-in user.

**Return type** List[*Karma*]

### Karma

The `Karma` model represents items in a `KarmaList` and contains information about the subreddit the karma was obtained on, as well as the amount of link and comment karma.

**class apraw.models.Karma**(*reddit: Reddit*, *data: Dict*)

A model representing subreddit karma.

**reddit: Reddit** The `Reddit` instance with which requests are made.

**data: Dict** The data obtained from the /about endpoint.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `sr` | The name of the subreddit the karma was obtained on |
| `comment_karma` | The amount of karma obtained on the subreddit. |
| `link_karma` | The amount of link karma obtained on the subreddit. |

`async subreddit()`

Retrieve the subreddit on which the karma was obtained.

**Returns** **subreddit** – The subreddit on which the karma was obtained.

**Return type** *Subreddit*

**Contents**

`class apraw.Reddit`(*praw_key: str = '', username: str = '', password: str = '', client_id: str = '', client_secret: str = '', user_agent='aPRAW by Dan6erbond'*)

The Reddit instance with which root requests can be made.

**user: User** An instance of the logged-in Reddit user.

**comment_kind: str** The prefix that represents *`Comment`* in API responses, such as `t1`.

**account_kind: str** The prefix that represents *`Redditor`* in API responses, such as `t2`.

**link_kind: str** The prefix that represents *`Submission`* in API responses, such as `t3`.

**message_kind: str** The prefix that represents *`Message`* in API responses, such as `t4`.

**subreddit_kind: str** The prefix that represents *`Subreddit`* in API responses, such as `t5`.

**award_kind: str** The prefix that represents awards in API responses, such as `t6`.

**modaction_kind: str** The prefix that represents *`ModAction`* in API responses, such as `modaction`.

**listing_kind: str** The prefix that represents *`Listing`* in API responses, such as `listing`.

**wiki_revision_kind: str** The prefix that represents *`WikipageRevision`* in API responses, such as `WikiRevision`.

**wikipage_kind: str** The prefix that represents *`SubredditWikipage`* in API responses, such as `wikipage`.

**more_kind: str** The prefix that represents *`MoreComments`* in API responses, such as `more`.

**request_handler: RequestHandler** An instance of `RequestHandler` with which this Reddit instance will perform HTTP requests.

`async comment`(*id: str = '', url: str = ''*) → *apraw.models.reddit.comment.Comment*

Get a *Comment* object based on its ID or URL.

**Parameters**

- **`id`** (*`str`*) – The ID of a comment (with or without kind).
- **`url`** (*`str`*) – The URL of a comment.

**Returns** **comment** – The requested comment.

**Return type** *Comment*

`async delete`(**args, **kwargs*) → Any

Perform an HTTP DELETE request on the Reddit API.

**Parameters**

- **`endpoint`** (*`str`*) – The endpoint to be appended after the base URL (https://oauth.reddit.com/).
- **`url`** (*`str`*) – The direct URL to perform the request on.

- **kwargs** – Query parameters to be appended after the URL.

**Returns** **resp** – The response JSON data.

**Return type** Any

`async get`(**args*, ***kwargs*) → Any

Perform an HTTP GET request on the Reddit API.

**Parameters**

- **endpoint** (*str*) – The endpoint to be appended after the base URL (https://oauth.reddit.com/).
- **kwargs** – Query parameters to be appended after the URL.

**Returns** **resp** – The response JSON data.

**Return type** Any

`async get_listing`(*endpoint: str*, *subreddit: Optional[apraw.models.subreddit.subreddit.Subreddit] = None*, *kind_filter: Optional[List[str]] = None*, ***kwargs*) → *apraw.models.reddit.listing.Listing*

Retrieve a listing from an endpoint.

**Parameters**

- **endpoint** (*str*) – The endpoint to be appended after the base URL (https://oauth.reddit.com/).
- **subreddit** (`Subreddit`) – The subreddit to dependency inject into retrieved items when possible.
- **kind_filter** – Kinds to return if given, otherwise all are returned.
- **kwargs** (***Dict*) – Query parameters to be appended after the URL.

**Returns** **listing** – The listing containing all the endpoint's children.

**Return type** *Listing*

`info`(*id: str = ''*, *ids: List[str] = []*, *url: str = ''*)

Get a Reddit item based on its ID or URL.

**Parameters**

- **id** (*str*) – The item's ID.
- **ids** (*List[str]*) – Multiple IDs to fetch multiple items at once (max 100).
- **url** (*str*) – The item's URL.

**Yields**

- **comment** (*Comment*) – A *Comment* object.
- **submission** (*Submission*) – A *Submission* object.

`async message`(*to: Union[str, apraw.models.reddit.redditor.Redditor]*, *subject: str*, *text: str*, *from_sr: Union[str, apraw.models.subreddit.subreddit.Subreddit] = ''*) → bool

Message a Redditor or Subreddit.

**Parameters**

- **to** (*str or* `Redditor` *or* `Subreddit`) – The Redditor or Subreddit the message should be sent to.
- **subject** (*str*) – The subject of the message.

- **text** (*str*) – The text contents of the message.
- **from_sr** (*str or* Subreddit) – Optional if the message is being sent from a subreddit.

**Returns** **result** – The response JSON data.

**Return type** Dict

async **post**(**args*, ***kwargs*) → Any
Perform an HTTP POST request on the Reddit API.

**Parameters**

- **endpoint** (*str*) – The endpoint to be appended after the base URL (https://oauth.reddit.com/).
- **url** (*str*) – The direct URL to perform the request on.
- **data** – The data to add to the POST body.
- **kwargs** – Query parameters to be appended after the URL.

**Returns** **resp** – The response JSON data.

**Return type** Any

async **put**(**args*, ***kwargs*) → Any
Perform an HTTP PUT request on the Reddit API.

**Parameters**

- **endpoint** (*str*) – The endpoint to be appended after the base URL (https://oauth.reddit.com/).
- **url** (*str*) – The direct URL to perform the request on.
- **data** – The data to add to the POST body.
- **kwargs** – Query parameters to be appended after the URL.

**Returns** **resp** – The response JSON data.

**Return type** Any

async **redditor**(*username: str*) → *apraw.models.reddit.redditor.Redditor*
Get a *Redditor* object based the Redditor's username.

**Parameters** **username** (*str*) – The Redditor's username (without '/u/').

**Returns** **redditor** – The requested Redditor, returns None if not found.

**Return type** *Redditor* or None

async **submission**(*id: str = ''*, *url: str = ''*) → *apraw.models.reddit.submission.Submission*
Get a *Submission* object based on its ID or URL.

**Parameters**

- **id** (*str*) – The ID of a submission (with or without kind).
- **url** (*str*) – The URL of a submission.

**Returns** **submission** – The requested submission.

**Return type** *Submission*

async **subreddit**(*display_name: str*) → *apraw.models.subreddit.subreddit.Subreddit*
Get a *Subreddit* object according to the given name.

**Parameters** `display_name` (*str*) – The display name of the subreddit.

**Returns** **subreddit** – The subreddit if found.

**Return type** *Subreddit*

`subreddits`(**args*, ***kwargs*)
A `ListingGenerator` that returns newly created subreddits, which can be streamed using `reddit.subreddits.stream()`.

**Parameters** `kwargs` (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` that retrieves newly created subreddits.

**Return type** *ListingGenerator*

## 2.4 aPRAW Models

This section contains the documentation and API of the implemented aPRAW models.

### 2.4.1 Subreddit

This section contains the documentation and API of the subreddit models and helpers.

#### Subreddit

This section describes the usage and members of the Subreddit model.

A subreddit can be instantiated as follows:

```
sub = await reddit.subreddit("aprawtest")
```

**class** `apraw.models.Subreddit`(*reddit: Reddit*, *data: Dict = None*)
The model representing subreddits.

**kind: str** The item's kind / type.

**mod: SubredditModeration** Returns an instance of `SubredditModeration`.

**modmail: SubredditModmail** Returns an instance of `SubredditModmail`.

**wiki: SubredditWiki** Returns an instance of `SubredditWiki`.

**Examples**

To grab new submissions made on a subreddit:

```
sub = await reddit.subreddit("aprawtest")
async for submission in sub.new(): # use .new.stream() for endless polling
    print(submission.title, submission.body)
```

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| accounts_active_is_fuzzed | bool |
| accounts_active | null |
| active_user_count | The number of active users on the subreddit. |
| advertiser_category | string |
| all_original_content | Whether the subreddit requires all content to be OC. |
| allow_discovery | Whether the subreddit can be discovered. |
| allow_images | Whether images are allowed as submissions. |
| allow_videogifs | Whether GIFs are allowed as submissions. |
| allow_videos | Whether videos are allowed as submissions. |
| banner_background_color | The banner's background color if applicable, otherwise empty. |
| banner_background_image | A URL to the subreddit's banner image. |
| banner_img | A URL to the subreddit's banner image if applicable. |
| banner_size | The subreddit's banner size if applicable. |
| can_assign_link_flair | Whether submission flairs can be assigned. |
| can_assign_user_flair | Whether the user can assign their own flair on the subreddit. |
| collapse_deleted_comments | Whether deleted comments should be deleted by clients. |
| comment_score_hide_mins | The minimum comment score to hide. |
| community_icon | A URL to the subreddit's community icon if applicable. |
| created_utc | The date on which the subreddit was created in UTC datetime. |
| created | The time the subreddit was created on. |
| description_html | The subreddit's description as HTML. |
| description | The subreddit's short description. |
| disable_contributor_requests | bool |
| display_name_prefixed | The subreddit's display name prefixed with 'r/'. |
| display_name | The subreddit's display name. |
| emojis_custom_size | The custom size set for emojis. |
| emojis_enabled | Whether emojis are enabled on this subreddit. |
| free_form_reports | Whether it's possible to submit free form reports. |
| has_menu_widget | Whether the subreddit has menu widgets. |
| header_img | A URL to the subreddit's header image of applicable. |
| header_size | The subreddit's header size. |
| header_title | The subreddit's header title. |
| hide_ads | Whether ads are hidden on this subreddit. |
| icon_img | A URL to the subreddit's icon image of applicable. |
| icon_size | The subreddit's icon size. |
| id | The subreddit's ID. |
| is_enroled_in_new_modmail | Whether the subreddit is enrolled in new modmail. |
| key_color | string |
| lang | The subreddit's language. |
| link_flair_enabled | Whether link flairs have been enabled for the subreddit. |
| link_flair_position | The position of link flairs. |
| mobile_banner_size | A URL to the subreddit's mobile banner if applicable. |
| name | The subreddit's fullname (t5_ID). |
| notification_level | |
| original_content_tag_enabled | Whether the subreddit has the OC tag enabled. |
| over18 | Whether the subreddit is NSFW. |
| primary_color | The subreddit's primary color. |
| public_description_html | The subreddit's public description as HTML. |
| public_description | The subreddit's public description string. |
| public_traffic | bool |

continues on next page

Table 1 – continued from previous page

| Attribute | Description |
|---|---|
| quarantine | Whether the subreddit is quarantined. |
| restrict_commenting | Whether comments by users are restricted on the subreddit. |
| restrict_posting | Whether posts to the subreddit are restricted. |
| show_media_preview | Whether media previews should be displayed by clients. |
| show_media | |
| spoilers_enabled | Whether the spoiler tag is enabled on the subreddit. |
| submission_type | The types of allowed submissions. Default is "any". |
| submit_link_label | The subreddit's submit label if applicable. |
| submit_text_html | The HTML submit text if a custom one is set on the subreddit. |
| submit_text_label | The text used for the submit button. |
| submit_text | The markdown submit text if a custom one is set on the subreddit. |
| subreddit_type | The subreddit type, either "public", "restricted" or "private". |
| subscribers | The number of subreddit subscribers. |
| suggested_comment_sort | The suggested comment sort algorithm, can be `null`. |
| title | The subreddit's banner title. |
| url | The subreddit's display name prepended with "/r/". |
| user_can_flair_in_sr | Whether the user can assign custom flairs (nullable). |
| user_flair_background_color | The logged in user's flair background color if applicable. |
| user_flair_css_class | The logged in user's flair CSS class. |
| user_flair_enabled_in_sr | Whether the logged in user's subreddit flair is enabled. |
| user_flair_position | The position of user flairs on the subreddit (right or left). |
| user_flair_richtext | The logged in user's flair text if applicable. |
| user_flair_template_id | The logged in user's flair template ID if applicable. |
| user_flair_text_color | The logged in user's flair text color. |
| user_flair_text | The logged in user's flair text. |
| user_flair_type | The logged in user's flair type. |
| user_has_favorited | Whether the logged in user has favorited the subreddit. |
| user_is_banned | Whether the logged in user is banned from the subreddit. |
| user_is_contributor | Whether the logged in user has contributed to the subreddit. |
| user_is_moderator | Whether the logged in user is a moderator on the subreddit. |
| user_is_muted | Whether the logged in user has been muted by the subreddit. |
| user_is_subscriber | Whether the logged in user is subscribed to the subreddit. |
| user_sr_flair_enabled | Whether the logged in user's subreddit flair is enabled. |
| user_sr_theme_enabled | Whether the logged in user has enabled the custom subreddit theme. |
| videostream_links_count | The number of submissions with videostream links. |
| whitelist_status | |
| wiki_enabled | Whether the subreddit has the wiki enabled. |
| wls | null |

**comments**(**args*, ***kwargs*)

Returns an instance of *ListingGenerator* mapped to the comments endpoint.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.comments.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* kwargs.

**Returns** **generator** – A `ListingGenerator` mapped to the comments endpoint.

**Return type** *ListingGenerator*

**async fetch()**
Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The `Subreddit` model with updated data.

**Return type** *Subreddit*

**hot**(**args*, ***kwargs*)
Returns an instance of `ListingGenerator` mapped to the hot submissions endpoint.

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to the hot submissions endpoint.

**Return type** *ListingGenerator*

**async message**(*subject: str*, *text: str*, *from_sr: Union[str,* apraw.models.subreddit.subreddit.Subreddit*] = ''*) → Dict
Send a message to the subreddit.

**Parameters**

- **subject** (*str*) – The message subject.
- **text** (*str*) – The message contents as markdown.
- **from_sr** (*str or* `Subreddit`) – The subreddit the message is being sent from if applicable.

**Returns** **response** – The API response JSON as a dictionary.

**Return type** Dict

**moderators**(***kwargs*) → AsyncIterator[*apraw.models.subreddit.moderation.SubredditModerator*]
Yields all the subreddit's moderators.

**Parameters** **kwargs** (***Dict*) – The query parameters to be added to the GET request.

**Yields** **moderator** (*SubredditModerator*) – An instance of the moderators as `SubredditModerator`.

**new**(**args*, ***kwargs*)
Returns an instance of `ListingGenerator` mapped to the new submissions endpoint.

**Note:** This listing can be streamed doing the following:

```python
for comment in submissions.new.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to the new submissions endpoint.

**Return type** *ListingGenerator*

**async random()**
Retrieve a random submission from the subreddit.

**Returns** **submission** – A random submission from the subreddit.

**Return type** *Submission*

`rising`(**args*, ***kwargs*)

Returns an instance of `ListingGenerator` mapped to the rising submissions endpoint.

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to the rising submissions endpoint.

**Return type** *ListingGenerator*

**async submit**(*title: str*, *kind: SubmissionKind*, ***kwargs*) → *Submission*

Make a new post to the subreddit. If *kind* is SubmissionKind.LINK then *url* is expected to be a valid url, otherwise *text* is expected (and it can be markdown text)

**Parameters**

- **title** (*str*) – The post's title.
- **kind** (*SubmissionKind*) – The post's kind.
- **url** (*str*) – Optional, the url if kind is LINK.
- **text** (*str*) – Optional, the text body of the post.
- **nsfw** (*bool = False*) – If the post if nsfw or not.
- **resubmit** (*bool = False*) – If the post is a re-submit or not. Needs to be True if a link with the same URL has already been submitted to the specified subreddi
- **spoiler** (*bool = False*) – If the post is a spoiler or not.

`top`(**args*, ***kwargs*)

Returns an instance of `ListingGenerator` mapped to the top submissions endpoint.

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to the top submissions endpoint.

**Return type** *ListingGenerator*

### Subreddit Moderation

This section details the usage of models related to subreddit moderation.

**Contents**

### SubredditModerator

Subreddit moderators are usually retrieved as follows:

```
sub = await reddit.subreddit("aprawtest")
moderators = []
async for moderator in sub.moderators():
    moderators.append(str(moderator))
```

**class** apraw.models.**SubredditModerator**(*reddit: Reddit*, *data: Dict*)

The model representing subreddit moderators. Redditors can be retrieved via *redditor()*.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *aPRAWBase* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `added` | The parsed UTC date on which the moderator was added. |
| `author_flair_css_class` | The moderator's flair CSS class in the respective subreddit. |
| `author_flair_text` | The moderator's flair text in the respective subreddit. |
| `date` | The UTC timestamp on which the moderator was added. |
| `id` | The Redditor's fullname (t2_ID). |
| `mod_permissions` | A list of all the moderator permissions or `["all"]`. |
| `name` | The Redditor's name. |

**__str__**()

Returns the Redditor's name.

**Returns** **name** – The Redditor's name.

**Return type** str

async **fetch**()

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *aPRAWBase*

property **fullname**

Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

async **redditor**() → *apraw.models.reddit.redditor.Redditor*

Retrieve the Redditor this Moderator represents.

**Returns** **redditor** – The Redditor that is represented by this object.

**Return type** *Redditor*

### SubredditModeration

Items in the modqueue can be fetched using the `modqueue` listing:

```
sub = await reddit.subreddit("aprawtest")
async for item in sub.mod.modqueue(): # can also be streamed
    print(type(item))
    >>> apraw.models.Comment or apraw.models.Submission
```

**class** `apraw.models.SubredditModeration`(*reddit: Reddit*, *subreddit:* Subreddit)
A helper class for grabbing listings to Subreddit moderation items.

**edited**(**args*, ***kwargs*)
Returns an instance of *ListingGenerator* mapped to grab edited items.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.edited.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* `kwargs`.

**Returns** **generator** – A *ListingGenerator* mapped to grab edited items.

**Return type** *ListingGenerator*

**log**(**args*, ***kwargs*)
Returns an instance of *ListingGenerator* mapped to grab mod actions in the subreddit log.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.log.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* `kwargs`.

**Returns** **generator** – A *ListingGenerator* mapped to grab mod actions in the subreddit log.

**Return type** *ListingGenerator*

**modqueue**(**args*, ***kwargs*)
Returns an instance of *ListingGenerator* mapped to grab items in the modqueue.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.modqueue.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* `kwargs`.

**Returns** **generator** – A *ListingGenerator* mapped to grab items in the modqueue.

**Return type** *ListingGenerator*

`reports`(**args*, ***kwargs*)
Returns an instance of `ListingGenerator` mapped to grab reported items.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.reports.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to grab reported items.

**Return type** *ListingGenerator*

async `settings`() → *apraw.models.subreddit.settings.SubredditSettings*
Retrieve the settings for the subreddit this helper works for.

**Returns** **settings** – The subreddit's settings with their data prefetched.

**Return type** *SubredditSettings*

`spam`(**args*, ***kwargs*)
Returns an instance of `ListingGenerator` mapped to grab items marked as spam.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.spam.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to grab items marked as spam.

**Return type** *ListingGenerator*

`unmoderated`(**args*, ***kwargs*)
Returns an instance of `ListingGenerator` mapped to grab unmoderated items.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.mod.unmoderated.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to grab unmoderated items.

**Return type** *ListingGenerator*

### SubredditSettings

**class** `apraw.models.SubredditSettings`(*reddit: Reddit*, *data: Dict[str, Any]*, *subreddit:* Subreddit = *None*)

A model representing subreddit settings.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *aPRAWBase* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `all_original_content` | Whether the subreddit only allows original content. |
| `allow_chat_post_creation` | Whether the subreddit allows chat post creation. |
| `allow_discovery` | Whether this subreddit can be discovered through the recommendations. |
| `allow_galleries` | Whether this subreddit allows submissions with galleries. |
| `allow_images` | Whether this subreddit allows image posts. |
| `allow_polls` | Whether this subreddit allows poll posts. |
| `allow_post_crossposts` | Whether this subreddit allows crossposts. |
| `allow_videos` | Whether this subreddit allows video submissions. |
| `collapse_deleted_comments` | Whether deleted comments in threads should be automatically collapsed. |
| `comment_score_hide_mins` | The comment score below which comments should be hidden. |
| `content_options` | `string` |
| `crowd_control_chat_level` | `int` |
| `crowd_control_level` | `int` |
| `crowd_control_mode` | `bool` |
| `default_set` | `bool` |
| `description` | The subreddit's short description. |
| `disable_contributor_requests` | `bool` |
| `domain` | `None` |
| `exclude_banned_modqueue` | Whether banned users should be excluded from the modqueue. |
| `free_form_reports` | Whether users can submit custom text reports. |
| `header_hover_text` | The hover text for the subreddit's header. |
| `hide_ads` | Whether ads should be hidden on this subreddit. |
| `key_color` | `string` |
| `language` | The subreddit's default language as a language code (i.e. "en" for English). |
| `original_content_tag_enabled` | Whether the subreddit has the OC tag enabled. |
| `over_18` | Whether this subreddit is marked NSFW. |
| `public_description` | The subreddit's public description. |
| `public_traffic` | `bool` |
| `restrict_commenting` | Whether comments are restricted on the subreddit. |
| `restrict_posting` | Whether posts are restricted on the subreddit. |
| `show_media_preview` | Whether media previews should be displayed by clients. |
| `show_media` | `bool` |
| `spam_comments` | The comment spam filter's setting, either "low", "medium" or "high". |
| `spam_links` | The link spam filter's setting, either "low", "medium" or "high". |
| `spam_selfposts` | The selfpost spam filter's setting, either "low", "medium" or "high". |
| `spoilers_enabled` | Whether the spoiler marker has been enabled on this subreddit. |
| `submit_link_label` | The submit button's label. |
| `submit_text_label` | The submit text's label. |
| `submit_text` | `string` |
| `subreddit_id` | The ID of the subreddit with the prepended kind i.e. `t5_`. |
| `subreddit_type` | One of "public", "private" or "restricted". |

continues on next page

Table 2 – continued from previous page

| Attribute | Description |
|---|---|
| `suggested_comment_sort` | The default comment sort for submissions. |
| `title` | The subreddit's name. |
| `toxicity_threshold_chat_level` | `int` |
| `welcome_message_enabled` | Whether the subreddit has enabled welcome messages. |
| `welcome_message_text` | The welcome message's text of this subreddit. |
| `wiki_edit_age` | The minimum account age requirement for wiki editors. |
| `wiki_edit_karma` | The minimum account karma requirement for wiki editors. |
| `wikimode` | The mode the wiki is in e.g. "modonly". |

**async fetch**() → *apraw.models.subreddit.settings.SubredditSettings*

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The `SubredditSettings` model with updated data.

**Return type** *SubredditSettings*

**property fullname**

Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

**async subreddit**() → *Subreddit*

Retrieve the subreddit this item was made in as a `Subreddit`.

**Returns** **subreddit** – The subreddit this item was made in.

**Return type** *Subreddit*

## ModAction

**class** apraw.models.**ModAction**(*reddit*, *data*, *subreddit=None*)

A model representing mod actions taken on specific items.

**reddit: Reddit** The `Reddit` instance with which requests are made.

**data: Dict** The data obtained from the /about endpoint.

**kind: str** The item's kind / type.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `action` | The type of action performed. |
| `created_utc` | The parsed UTC datetime of when the action was performed. |
| `description` | The description added to the action if applicable. |
| `details` | The details of the action performed. |
| `id` | The ID of the mod action prepended with "*ModAction*". |
| `mod_id36` | The ID36 of the moderator who performed the action. |
| `mod` | The username of the moderator who performed the action. |
| `sr_id36` | The ID36 of the subreddit the action was performed on. |
| `subreddit_name_prefixed` | The name of the subreddit the action was performed on prefixed with "r/". |
| `subreddit` | The name of the subreddit the action was performed on. |
| `target_author` | The author of the target item if applicable. |
| `target_body` | The body of the target item if applicable. |
| `target_fullname` | The id of the target with its kind prepended. (e.g. "t3_d5229o") |
| `target_permalink` | The target of the comment or submission if applicable. |
| `target_title` | The title of the submission if applicable. |

`async mod()` → *apraw.models.reddit.redditor.Redditor*

Returns the Redditor who performed this action.

**Returns** **redditor** – The Redditor who performed this action.

**Return type** *Redditor*

### Subreddit Banned

This section details the usage of models related to banned users of a subreddit.

**Contents**

#### SubredditBanned

A helper class to aid in interacting with a subreddit's banned users.

#### BannedUser

Banned users can be fetched doing the following:

```
sub = await reddit.subreddit("aprawtest")
async for item in sub.banned(): # can also be streamed
    print(type(item))
    >>> apraw.models.BannedUser
```

`class apraw.models.BannedUser`(*reddit: Reddit*, *data: Dict*, *subreddit:* Subreddit)

The model representing banned users on a subreddit. The Redditor can be retrieved via *`redditor()`*.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `banned` | The parsed UTC date on which the user was banned. |
| `date` | The UTC timestamp on which the user was banned. |
| `days_left` | The number of days left for the ban. `0` if permanent. |
| `id` | The Redditor's fullname (t2_ID). |
| `name` | The Redditor's name. |
| `note` | The ban note added by the subreddit moderators. |
| `rel_id` | `str` |

`__str__()`
Returns the Redditor's name.

**Returns** **name** – The Redditor's name.

**Return type** str

`async fetch()`
Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *aPRAWBase*

`property fullname`
Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

`async redditor()` → *apraw.models.reddit.redditor.Redditor*
Retrieve the Redditor this Moderator represents.

**Returns** **redditor** – The Redditor that is represented by this object.

**Return type** *Redditor*

## Subreddit Modmail

This section details the usage of models related to subreddit modmail.

**Contents**

### ModmailMessage

**class** apraw.models.**ModmailMessage**(*reddit: Reddit*, *data: Dict*, *conversation: apraw.models.subreddit.modmail.ModmailConversation*)

The model for modmail messages.

**conversation: ModmailConversation** The `ModmailConversation` instance this message belongs to.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `id` | The ID of this message. |
| `body` | The HTML body of this message. |
| `body_markdown` | The raw body of this message. |
| `body_md` | An alias to `body_markdown`. |
| `is_internal` | Whether the message was sent internally. |
| `date` | The datetime string on which the message was sent. |

**async author**() → Optional[*Redditor*]

Retrieve the author of this message as a `Redditor`.

**Returns** **author** – The author of this modmail message if they haven't been deleted yet.

**Return type** *Redditor* or None

### SubredditModmail

**class** apraw.models.**SubredditModmail**(*reddit: Reddit*, *subreddit:* Subreddit)

Helper class to aid in retrieving subreddit modmail.

**async __call__**(*id: str*, *mark_read=False*) → *apraw.models.subreddit.modmail.ModmailConversation*

Fetch a `ModmailConversation` by its ID.

**Parameters** **id** (*str*) – The conversation's ID.

**Returns** **conversation** – The conversation requested if it exists.

**Return type** *ModmailConversation*

**conversations**() → *apraw.models.subreddit.modmail.ModmailConversation*

Retrieve a list of modmail conversations.

**Yields** **conversation** (*ModmailConversation*) – A modmail conversation held in the subreddit.

### ModmailConversation

**class** `apraw.models.ModmailConversation`(*reddit: Reddit*, *data: Dict*, *owner:* Subreddit *= None*)

The model for modmail conversations.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *`aPRAWBase`* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `authors` | A list of dictionaries containing authors by name with additional meta information such as `isMod`, `isAdmin`, `isOp`, `isParticipant`, `isHidden`, `id`, `isDeleted`. |
| `id` | The ID of this conversation. |
| `is_auto` | `bool` |
| `is_highlighted` | Whether the conversation has been highlighted. |
| `is_internal` | Whether it's an internal mod conversation. |
| `is_repliable` | Whether the conversation can be replied to. |
| `last_mod_update` | A timestamp of the last moderator update or `None`. |
| `last_unread` | `None` |
| `last_updated` | A timestamp of the last update made overall. |
| `last_user_update` | A timestamp of the last user update or `None`. |
| `num_messages` | The number of messages in this conversation. |
| `obj_ids` | A list of dictionaries containing the objects with their IDs and keys. |
| `owner` | A dictionary describing the subreddit this conversation is held in. |
| `participant` | `Dict` |
| `state` | `int` |
| `subject` | The subject of this conversation. |

**async** `archive()`

Archive the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

**async** `fetch`(*mark_read=False*)

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

**async** `highlight()`

Highlight the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

`messages()` → *apraw.models.subreddit.modmail.ModmailMessage*

Retrieve the messages sent in this conversation.

**Yields** **message** (*ModmailMessage*) – A message sent in this conversation.

**async** `mute()`

Mute the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

`async owner()` → *Subreddit*

Retrieve the owner subreddit of this conversation.

**Returns** **owner** – The subreddit this conversation was held in.

**Return type** *Subreddit*

`async remove_highlight()`

Remove the highlight from the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

`async reply(`*body: str*, *author_hidden: bool = False*, *internal: bool = False*`)`

Reply to the modmail conversation.

**Parameters**

- **body** (*str*) – The markdown reply body.
- **author_hidden** (*bool*) – Whether the author of this reply should be hidden.
- **internal** (*bool*) – Whether the reply is internal.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

`async unarchive()`

Unarchive the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

`async unmute()`

Unmute the modmail conversation.

**Returns** **self** – The updated model.

**Return type** *ModmailConversation*

### Subreddit Wiki

This section details the usage of models related to subreddit wiki.

**Contents**

### SubredditWiki

**class** apraw.models.**SubredditWiki**(*reddit: Reddit*, *subreddit: Subreddit*)

A helper class to aid in retrieving subreddit wiki pages, revisions as well as creating items.

**async** **__call__**() → List[str]

Retrieve a list of the available wikipages.

**Returns** **pages** – A list of all the wikipages in the subreddit by name.

**Return type** List[str]

**async** **create**(*page: str*, *content_md: str = ''*, *reason: str = ''*) → *apraw.models.subreddit.wiki.SubredditWikipage*

Create a new wikipage on the subreddit.

**Parameters**

- **page** (*str*) – The wikipage's name.
- **content_md** (*str*) – The wikipage's content as a markdown string.
- **reason** (*str*) – An optional string detailing the reason for the creation of this wikipage.

**Returns** **wikipage** – The newly created wikipage.

**Return type** *SubredditWikipage*

**async** **page**(*page: str*) → *apraw.models.subreddit.wiki.SubredditWikipage*

Retrieve a specific *SubredditWikipage* by its name.

**Parameters** **page** (*str*) – The wikipage's name which can be retrieved using the list from *__call__()*.

**Returns** **wikipage** – The requested wikipage if it exists.

**Return type** *SubredditWikipage*

**revisions**(**args*, ***kwargs*)

Returns an instance of *ListingGenerator* mapped to recent wikipage revisions.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.wiki.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* kwargs.

**Returns** **generator** – A *ListingGenerator* mapped to recent wikipage revisions.

**Return type** *ListingGenerator*

### SubredditWikipage

`class apraw.models.SubredditWikipage`(*name: str*, *reddit: Reddit*, *subreddit:* Subreddit, *data: Dict = None*)

The model that represents Subreddit wikipages.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `content_html` | The content's of the wikipage as an HTML string. |
| `content_md` | The content's of the wikipage formatted as a markdown string. |
| `may_revise` | `bool` |
| `name` | The wikipage's name. |
| `reason` | The reason text for the wikipage's current revision. |
| `revision_by` | The author of the wikipage's current revision. |
| `revision_date` | The date on which the current revision was made. |
| `revision_id` | The ID of the wikipage's current revision. |

`async add_editor`(*username: str*) → Union[bool, Any]

Add a Redditor to the editors of this wikipage.

**Parameters** **username** (*str*) – The Redditor's username without the prefix.

**Returns** **res** – `True` if the request was successful, otherwise the response's raw data.

**Return type** bool or Any

`async del_editor`(*username: str*) → Union[bool, Any]

Remove a Redditor from the editors of this wikipage.

**Parameters** **username** (*str*) – The Redditor's username without the prefix.

**Returns** **res** – `True` if the request was successful, otherwise the response's raw data.

**Return type** bool or Any

`async edit`(*content_md: str*, *reason: Optional[str] = ''*) → Union[bool, Any]

Edit a wikipage's markdown contents.

**Parameters**

- **content_md** (*str*) – The new wikipage's content as a markdown string.
- **reason** (*Optional[str]*) – An optional reason for this edit.

**Returns** **res** – `True` if the request was successful, otherwise the response's raw data.

**Return type** bool or Any

`async hide`(*revision: Union[str,* apraw.models.subreddit.wiki.WikipageRevision*]*)

Hide a wikipage revision from the public history.

**Parameters** **revision** (*str or* WikipageRevision) – The wikipage revision either as a `WikipageRevision` or its ID string.

**Returns** **res** – `True` if the request was successful, otherwise the response's raw data.

**Return type** bool or Any

`async revert`(*revision: Union[str,* apraw.models.subreddit.wiki.WikipageRevision*]*) → Union[bool, Any]

Revert a wikipage to its previous revision.

**Parameters** **revision** (*str or WikipageRevision*) – The wikipage revision either as a *WikipageRevision* or its ID string.

**Returns** **res** – `True` if the request was successful, otherwise the response's raw data.

**Return type** bool or Any

**revisions**(**args*, ***kwargs*)
Returns an instance of *ListingGenerator* mapped to fetch specific wikipage revisions.

**Note:** This listing can be streamed doing the following:

```
for comment in subreddit.wiki.page("test").stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – *ListingGenerator* `kwargs`.

**Returns** **generator** – A *ListingGenerator* mapped to fetch specific wikipage revisions.

**Return type** *ListingGenerator*

### WikipageRevision

**class** apraw.models.**WikipageRevision**(*reddit: Reddit*, *data: Dict = None*)
The model that represents wikipage revisions.

**author: Redditor** The Redditor that made this revision.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *aPRAWBase* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `timestamp` | A timestamp of when the revision was made. |
| `page` | The name of the page the revision addresses. |
| `revision_hidden` | Whether the revision has been hidden by the editors. |
| `reason` | The reason string for this revision if available. |
| `id` | The ID of this revision. |

### Removal Reasons

This section details the usage of models that aid in fetching and managing subreddit removal reasons.

**Contents**

### SubredditRemovalReasons

**class** apraw.models.**SubredditRemovalReasons**(*reddit: Reddit*, *subreddit:* Subreddit)

A helper to aid in retrieving and adding removal reasons to a subreddit.

**async add**(*title: str*, *message: str*) → *apraw.models.subreddit.removal_reasons.SubredditRemovalReason*

Add a removal reason to the subreddit's list.

**Parameters**

- `title` (*str*) – The title under which this removal reason is saved.
- `message` (*str*) – The message that is sent to author's when the removal reason is used.

**Returns** **reason** – The newly created, and fetched, removal reason.

**Return type** *SubredditRemovalReason*

**async get**(*item: Union[int, str]*) → *apraw.models.subreddit.removal_reasons.SubredditRemovalReason*

Retrieve a removal reason based on its ID or index.

**Parameters** `item` (*int or str*) – The item's ID or index.

**Returns** **reason** – The removal reason that was found in the list.

**Return type** *SubredditRemovalReason*

**Raises**

- `StopIteration` – If no removal reason by the given ID was found.
- `IndexError` – If the index given doesn't exist in the list of removal reasons.

### SubredditRemovalReason

**class** apraw.models.**SubredditRemovalReason**(*reddit: Reddit*, *subreddit:* Subreddit, *data: Dict*)

The model representing subreddits.

**url: str** The API URL to this specific removal reason.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `message` | The message for this removal reason that is sent to authors. |
| `id` | The ID of this removal reason. |
| `title` | The title of this removal reason in the subreddit. |

**async delete**() → Any

Delete this removal reason from the subreddit.

**Returns** **response** – The API endpoint raw response.

**Return type** Any

**async fetch**()

Fetch the data for this removal reason. The `aPRAWBase` class will automatically update and/or add members returned by the API.

**async update**(*title: Optional[str] = None*, *message: Optional[str] = None*) → Any
Update the title and/or message of this removal reason.

**Parameters**

- **title** (*Optional[str]*) – The updated title for this removal reason. If none is specified the original title will be reused.
- **message** (*Optional[str]*) – The updated message for this removal reason. If none is specified the original message will be reused.

**Returns** **response** – The API endpoint raw response.

**Return type** Any

## 2.4.2 Submission

This section contains the documentation and API of the submission model and its moderation helper class.

### Submission

A Submission can either be instantiated by using its ID, or by going through subreddits:

```
submission = await reddit.submission("h7mna9")

sub = await reddit.redditor("aprawbot")
async for submission in sub.new():
    print(submission)
```

**class** apraw.models.**Submission**(*reddit: Reddit*, *data: Dict*, *subreddit:* Subreddit *= None*, *author:* apraw.models.reddit.redditor.Redditor *= None*)
The model representing submissions.

**reddit: Reddit** The *Reddit* instance with which requests are made.

**data: Dict** The data obtained from the /about endpoint.

**mod: SubmissionModeration** The *SubmissionModeration* instance to aid in moderating the submission.

**kind: str** The item's kind / type.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *aPRAWBase* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| all_awardings | A list of the awardings on the submission. |
| allow_live_comments | Whether live comments have been enabled on this submission. |
| approved_at_utc | The UTC timestamp of when the submission was approved. |
| approved_by | The user that approved the submission. |
| approved | Whether the submission has been approved by the moderators of the subreddit. |
| archived | Whether the submission has been archived by Reddit. |
| author_flair_background_color | The submission author's flair background color. |
| author_flair_css_class | The submission's author flair CSS class. |
| author_flair_richtext | The submission's author flair text. |

continues on next page

Table 3 – continued from previous page

| Attribute | Description |
|---|---|
| `author_flair_template_id` | The submission author's flair template ID if applicable. |
| `author_flair_text_color` | The submission's author flair text color if applicable. |
| `author_flair_text` | The author's flair text if applicable. |
| `author_flair_type` | The type of flair used by the submission's author. |
| `author_fullname` | The author of the submission prepended with `t2_`. |
| `author_patreon_flair` | The submission's author Patreon flair. |
| `author` | The name of the submission's Redditor. |
| `banned_at_utc` | The UTC timestamp at which the author was banned. |
| `banned_by` | `null` |
| `can_gild` | Whether the logged-in user can gild the submission. |
| `can_mod_post` | Whether the logged-in user can modify the post. |
| `category` | The submission's category. |
| `clicked` | Whether the submission has been clicked by the logged-in user previously. |
| `content_categories` | The content categories assigned to the submission. |
| `contest_mode` | Whether the moderators of the subreddit have enabled contest mode on the submission. |
| `created_utc` | The parsed UTC `datetime` on which the submission was made. |
| `created` | The timestamp of when the submission was posted. |
| `discussion_type` | `null` |
| `distinguished` | The type of distinguishment on the submission. |
| `domain` | The domain of the submission. |
| `downs` | The number of downvotes on the submission. |
| `edited` | Whether the submission has been edited by its author. |
| `gilded` | The number of awards this submission has received. |
| `gildings` | The gild awards the submission has received. |
| `hidden` | Whether the submission has been hidden by the logged-in user. |
| `hide_score` | Whether clients should hide the score from users. |
| `id` | The submission's ID. |
| `ignore_reports` | Whether reports should be ignored on this submission.`` |
| `is_crosspostable` | Whether the submission can be crossposted to other subreddits. |
| `is_meta` | Whether the submission is a meta post. |
| `is_original_content` | Whether the submission has been marked as original content. |
| `is_reddit_media_domain` | Whether the media has been uploaded to Reddit. |
| `is_robot_indexable` | Whether the submission can be indexed by robots. |
| `is_self` | Whether the submission is a self post. |
| `is_video` | Whether the submission is a video post. |
| `likes` | `bool` |
| `link_flair_background_color` | The submission's flair background color. |
| `link_flair_css_class` | The CSS class applied on the submission's flair if applicable. |
| `link_flair_richtext` | The submission's flair text if applicable. |
| `link_flair_template_id` | The submission's flair template ID if applicable. |
| `link_flair_text_color` | The submission's flair text color if applicable. |
| `link_flair_text` | The submission's flair text. |
| `link_flair_type` | The type of flair applied to the submission. |
| `locked` | Whether the submission has been locked by the subreddit moderators. |
| `media_embed` | `Dict` |
| `media_only` | Whether the submission only consists of media. |
| `media` | `null` |
| `mod_note` | Moderator notes added to the submission. |
| `mod_reason_by` | The moderator who added the removal reason if applicable. |

continues on next page

Table 3 – continued from previous page

| Attribute | Description |
|---|---|
| `mod_reason_title` | The reason the submission has been removed by moderators if applicable. |
| `mod_reports` | A list of moderator reports on the submission. |
| `name` | The ID of the submission prepended with `t3_`. |
| `no_follow` | `bool` |
| `num_comments` | The number of comments on the submission. |
| `num_crossposts` | The number of times the submission has been crossposted. |
| `num_reports` | The number of reports on the submission. |
| `over_18` | Whether the submission has been marked as NSFW. |
| `parent_whitelist_status` | `null` |
| `permalink` | The submission's permalink. |
| `pinned` | Whether the submission has been pinned on the subreddit. |
| `pwls` | `null` |
| `quarantine` | Whether the submission was posted in a quarantined subreddit. |
| `removal_reason` | The submission's removal reason if applicable. |
| `removed` | Whether the submission has been removed by the subreddit moderators. |
| `report_reasons` | A list of report reasons on the submission. |
| `saved` | Whether the submission has been saved by the logged-in user. |
| `score` | The overall submission vote score. |
| `secure_media_embed` | `Dict` |
| `secure_media` | `null` |
| `selftext_html` | The submission text as HTML. |
| `selftext` | The submission's selftext. |
| `send_replies` | Whether the author of the submission will receive reply notifications. |
| `spam` | Whether the submission has been marked as spam. |
| `spoiler` | Whether the submission contains a spoiler. |
| `stickied` | Whether the submission is stickied on the subreddit. |
| `subreddit_id` | The subreddit's ID prepended with `t5_`. |
| `subreddit_name_prefixed` | The name of the subreddit the submission was posted on, prefixed with "r/". |
| `subreddit_subscribers` | The number of subscribers to the submission's subreddit. |
| `subreddit_type` | The type of the subreddit the submission was posted on (public, restricted, private). |
| `subreddit` | The name of the subreddit on which the submission was posted. |
| `suggested_sort` | The suggested sort method for comments. |
| `thumbnail_height` | The height of the submission's thumbnail if applicable. |
| `thumbnail_width` | The width of the submission's thumbnail if applicable. |
| `thumbnail` | A URL to the submission's thumbnail if applicable. |
| `title` | The submission's title. |
| `total_awards_received` | The number of awards on the submission. |
| `ups` | The number of upvotes on the submission. |
| `url` | The full URL of the submission. |
| `user_reports` | A list of the user reports on the submission. |
| `view_count` | The number of views on the submission. |
| `visited` | Whether the logged-in user has visited the submission previously. |
| `whitelist_status` | `null` |
| `wls` | `null` |

**Note:** Many of these attributes are only available if the logged-in user has moderator access to the item.

**async author()** → *apraw.models.reddit.redditor.Redditor*

Retrieve the item's author as a `Redditor`.

**Returns** **author** – The item's author.

**Return type** *Redditor*

`async clear_vote()`
Clear user up- and downvotes on the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async comment(text: str)` → Union[*Comment*, *Message*]
Reply to the item.

**Returns** **reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

`async delete()`
Delete the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async downvote()`
Downvote the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async fetch()`
Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *Submission*

`property fullname`
Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

`async hide()`
Hide the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async mark_nsfw()`
Mark the item as NSFW.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async mark_spoiler()`
Mark the item as a spoiler.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async reply**(*text: str*) → Union[*Comment*, *Message*]

Reply to the item.

**Returns** **reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

**async save**(*category: str = ''*)

Save the item in a category.

**Parameters** **category** (`str, optional`) – The category name.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async subreddit**() → *Subreddit*

Retrieve the subreddit this item was made in as a `Subreddit`.

**Returns** **subreddit** – The subreddit this item was made in.

**Return type** *Subreddit*

**async unhide**()

Unhide the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async unmark_nsfw**()

Unmark the item as NSFW.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async unmark_spoiler**()

Unmark the item as a spoiler.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async unsave**()

Unsave the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async upvote**()

Upvote the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

### Submission Moderation

**class** apraw.models.**SubmissionModeration**(*reddit: Reddit*, *submission: apraw.models.reddit.submission.Submission*)

A helper class to moderate submissions.

**async approve**()

Approve the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async distinguish**(*how: apraw.models.enums.distinguishment_option.DistinguishmentOption = 'yes'*, *sticky: bool = False*)

Distinguish the Reddit item.

**Parameters**

- **how** (`DistinguishmentOption`) – The type of distinguishment to be added to the item.
- **sticky** (*bool, optional*) – Whether the item should be stickied.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async flair**(*text: str*, *css_class: str = ''*)

Flair a submission.

**Parameters**

- **text** (*str*) – The flair text string no longer than 64 characters.
- **css_class** (*str*) – A valid subreddit image name.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**property fullname: str**

Retrieve the fullname of the item this helper performs requests for.

**Returns** **fullname** – The ID prepended with the kind of the item this helper belongs to.

**Return type** str

**async ignore_reports**()

Ignore reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async lock**()

Lock the item from further replies.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async mark_nsfw**()

Mark the item as NSFW.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async mark_spoiler()`
Mark the item as a spoiler.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async remove`(*spam: bool = False*, *mod_note: Optional[str] = ''*, *reason: Optional[Union[str, apraw.models.subreddit.removal_reasons.SubredditRemovalReason]] = None*)
Remove the Reddit item.

**Parameters**

- **spam** (*bool*) – When `True`, use the removal to help train the Subreddit's spam filter (default: `False`).
- **mod_note** (*Optional[str]*) – A message for the other moderators.
- **reason** (*str or SubredditRemovalReason*) – The removal reason ID or a subreddit removal reason to add.

**Returns** **resp** – The API response JSON or a tuple of dictionaries if a removal reason / mod note was added as well.

**Return type** Dict or Tuple

`async sticky`(*position: int = 1*, *to_profile: bool = False*)
Sticky a submission in its subreddit.

**Parameters**

- **position** (*int*) – The "slot" the submission will be stickied to.
- **to_profile** (*bool*) – Whether the submission will be stickied to the user profile.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async undistinguish()`
Undistinguish the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unignore_reports()`
Unignore previously ignored reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unlock()`
Unlock the item from further replies.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unmark_nsfw()`
Unmark the item as NSFW.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async unmark_spoiler()**
Unmark the item as a spoiler.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async unsticky**(*to_profile: bool = False*)
Unsticky a submission from its subreddit.

**Parameters** **to_profile** (*bool*) – Whether the submission will be unstickied from the user profile.

**Returns** **resp** – The API response JSON.

**Return type** Dict

## 2.4.3 Comment

This section contains the documentation and API of the comment model and its moderation helper class.

### Comment

Besides retrieving comments similarly to submissions using their ID or fetching them through a subreddit's listings, comments can be obtained from the submission they were made in like so:

```
submission = await reddit.submission("h7mna9")

async for comment in submission.comments():
    print(comment)
```

**class** apraw.models.**Comment**(*reddit: Reddit*, *data: Dict*, *submission: Submission = None*, *author: apraw.models.reddit.redditor.Redditor = None*, *subreddit: Subreddit = None*, *replies: Union[CommentForest, List] = None*)
The model representing comments.

**mod: CommentModeration** The *CommentModeration* instance to aid in moderating the comment.

**kind: str** The item's kind / type.

**url: str** The URL pointing to this comment.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the *aPRAWBase* class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| all_awardings | A list of awardings added to the comment. |
| approved_at_utc | The UTC timestamp at which the comment was approved by the moderators. |
| approved_by | The moderator who approved this comment if applicable. |
| approved | Whether the comment has been approved by the moderators. |
| archived | Whether the comment has been archived. |
| author_flair_background_color | The comment author's flair background color if applicable. |
| author_flair_css_class | The comment author's flair CSS class if applicable. |
| author_flair_richtext | The comment author's flair text if applicable. |

continues on next page

Table 4 – continued from previous page

| Attribute | Description |
|---|---|
| `author_flair_template_id` | The comment author's flair template ID if applicable. |
| `author_flair_text_color` | The comment author's flair text color if applicable. |
| `author_flair_text` | The comment author's flair text if applicable. |
| `author_flair_type` | The comment author's flair type if applicable. |
| `author_fullname` | The comment author's ID prepended with `t2_`. |
| `author_patreon_flair` | The comment author's Patreon flair if applicable. |
| `author` | The comment author's username. |
| `banned_at_utc` | `None` |
| `banned_by` | `None` |
| `body_html` | The HTML version of the comment's body. |
| `body` | The comment's markdown body. |
| `can_gild` | Whether the logged-in user can gild the comment. |
| `can_mod_post` | `bool` |
| `collapsed_reason` | `None` |
| `collapsed` | Whether the comment should be collapsed by clients. |
| `controversiality` | A score on the comment's controversiality based on its up- and downvotes. |
| `created_utc` | The parsed UTC `datetime` on which the comment was made. |
| `created` | A timestamp on which the comment was created. |
| `distinguished` | The type of distinguishment the comment hsa received. |
| `downs` | The number of downvotes the comment has received. |
| `edited` | Whether the comment has been edited from its original state. |
| `gilded` | The number of awards this comment has received. |
| `gildings` | A dictionary of gilds the comment has received. |
| `id` | The comment's ID. |
| `ignore_reports` | Whether reports should be ignored on this comment. |
| `is_submitter` | Whether the logged-in user is the submitter of this comment. |
| `likes` | The overall upvote score on this comment. |
| `link_author` | The username of the comment submission's author. |
| `link_id` | The ID of the submission this comment was made in. |
| `link_permalink` / `link_url` | A URL to the comment's submission. |
| `link_title` | The comment's submission title. |
| `locked` | Whether the comment has been locked by the moderators. |
| `mod_note` | Notes added to the comment by moderators if applicable. |
| `mod_reason_by` | The moderator who added a removal reason if applicable. |
| `mod_reason_title` | The mod reason's title if applicable. |
| `mod_reports` | A list of reports made on this comment filed by moderators. |
| `name` | The comment's ID prepended with `t1_`. |
| `no_follow` | `bool` |
| `num_comments` | The number of replies made in this submission. |
| `num_reports` | The number of reports on this comment. |
| `over_18` | Whether the comment has been marked NSFW. |
| `parent_id` | The comment's parent ID, either `link_id` or the ID of another comment. |
| `permalink` | The comment's permalink. |
| `quarantine` | `bool` |
| `removal_reason` | A removal reason set by moderators if applicable. |
| `removed` | Whether the comment has been removed by the moderators of the subreddit. |
| `replies` | A list of replies made under this comment, usually empty at first. |
| `report_reasons` | Report reasons added to the comment. |
| `saved` | Whether the logged-in user has saved this comment. |

continues on next page

Table 4 – continued from previous page

| Attribute | Description |
|---|---|
| `score_hidden` | Whether clients should hide the comment's score. |
| `score` | The overall upvote score on this comment. |
| `send_replies` | Whether the OP has enabled reply notifications. |
| `spam` | Whether the comment has been flagged as spam. |
| `stickied` | Whether the comment has been stickied by the moderators. |
| `subreddit_id` | The comment subreddit's ID prepended with `t5_`. |
| `subreddit_name_prefixed` | The comment's subreddit name prefixed with "r/". |
| `subreddit_type` | The type of the subreddit the submission was posted on (public, restricted, private). |
| `subreddit` | The name of the subreddit this comment was made in. |
| `total_awards_received` | The number of awards this comment has received. |
| `ups` | The number of upvotes this comment has received. |
| `user_reports` | A list of user reports filed for this comment. |

**Note:** Many of these attributes are only available if the logged-in user has moderator access to the item.

`async author()` → *apraw.models.reddit.redditor.Redditor*

Retrieve the item's author as a `Redditor`.

**Returns** **author** – The item's author.

**Return type** *Redditor*

`async clear_vote()`

Clear user up- and downvotes on the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async comment(text: str)` → Union[*Comment*, *Message*]

Reply to the item.

**Returns** **reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

`async delete()`

Delete the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async downvote()`

Downvote the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async fetch()`

Fetch this item's information from a suitable API endpoint.

**Returns** **update** – Whether ReactivePy attributes have been changed.

**Return type** bool

`property fullname`

Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

**async hide()**

Hide the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async link()** → *Submission*

Retrieve the submission this item belongs to as a `Submission`.

**Returns** **submission** – The item's parent submission.

**Return type** *Submission*

**async monitor**(*max_wait=16*)

Continuously fetch this comment's data to react to changes in the data.

This can be used in combination with the callbacks offered by ReactivePy to be notified on changes in specific fields on this comment. For more information on ReactivePy view the GitHub repo.

Callbacks can be assigned as follows:

```
async def on_change(*args):
    for arg in args:
        print(f"{arg.name} changed to {arg.value}.")
comment.on_change(on_change)
```

**Note:** Callbacks will work regardless of them being asynchronous or synchronous, as aPRAW's models use the `_async_bulk_update()` method offered by ReactivePy's interface.

**Parameters** **max_wait** (*int*) – The maximum amount of time to wait between requests if no updates were previously recognized.

**async reply**(*text: str*) → Union[*Comment*, *Message*]

Reply to the item.

**Returns** **reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

**async save**(*category: str = ''*)

Save the item in a category.

**Parameters** **category** (*str, optional*) – The category name.

**Returns** **resp** – The API response JSON.

**Return type** Dict

**async submission()** → *Submission*

Retrieve the submission this item belongs to as a `Submission`.

**Returns** **submission** – The item's parent submission.

**Return type** *Submission*

**async subreddit()** → *Subreddit*

Retrieve the subreddit this item was made in as a `Subreddit`.

**Returns** **subreddit** – The subreddit this item was made in.

**Return type** *Subreddit*

`async unhide()`

Unhide the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unsave()`

Unsave the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async upvote()`

Upvote the item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

### Comment Moderation

`class apraw.models.CommentModeration(`*reddit: Reddit*, *comment:* apraw.models.reddit.comment.Comment`)`

A helper class to moderate comments.

`async approve()`

Approve the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async distinguish(`*how:* apraw.models.enums.distinguishment_option.DistinguishmentOption = *'yes'*, *sticky: bool = False*`)`

Distinguish the Reddit item.

**Parameters**

- **how** (`DistinguishmentOption`) – The type of distinguishment to be added to the item.
- **sticky** (*bool, optional*) – Whether the item should be stickied.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`property fullname: str`

Retrieve the fullname of the item this helper performs requests for.

**Returns** **fullname** – The ID prepended with the kind of the item this helper belongs to.

**Return type** str

`async ignore_reports()`

Ignore reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async lock()`
Lock the item from further replies.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async remove`(*spam: bool = False*, *mod_note: Optional[str] = ''*, *reason: Optional[Union[str, apraw.models.subreddit.removal_reasons.SubredditRemovalReason]] = None*)
Remove the Reddit item.

**Parameters**

- **`spam`** (*`bool`*) – When `True`, use the removal to help train the Subreddit's spam filter (default: `False`).
- **`mod_note`** (*`Optional[str]`*) – A message for the other moderators.
- **`reason`** (*`str or SubredditRemovalReason`*) – The removal reason ID or a subreddit removal reason to add.

**Returns** **resp** – The API response JSON or a tuple of dictionaries if a removal reason / mod note was added as well.

**Return type** Dict or Tuple

`async show_comment()`
Mark a comment that it should not be collapsed because of crowd control.

The comment could still be collapsed for other reasons.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async undistinguish()`
Undistinguish the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unignore_reports()`
Unignore previously ignored reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async unlock()`
Unlock the item from further replies.

**Returns** **resp** – The API response JSON.

**Return type** Dict

### 2.4.4 Message

This section describes the usage and members of the Message model.

Messages are the private messages sent and received via the old Reddit private messaging system and are conventionally retrieved through the inbox:

```
async for message in reddit.user.inbox.unread():
    print(message)
```

**class** apraw.models.**Message**(*reddit: Reddit*, *data: Dict[str, Any]*)

The model representing comments.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `first_message` | The first message sent in the message thread if the current message wasn't the first. |
| `first_message_name` | The fullname of the first message in the message thread if applicable. |
| `subreddit` | The subreddit this conversation is being held in if applicable. |
| `likes` | `None` |
| `replies` | A list of all the message replies if applicable, otherwise an empty string. |
| `id` | The message ID. |
| `subject` | The subject of this message's thread. |
| `associated_awarding_id` | The ID of the associated awarding if the message was sent in the context of an award. |
| `score` | `0` |
| `author` | The username of the message's author. |
| `num_comments` | The number of comments in this message's thread. |
| `parent_id` | `None` |
| `subreddit_name_prefixed` | The prefixed name of the subreddit this conversation is being held in if applicable. |
| `new` | `bool` |
| `type` | `str` |
| `body` | The markdown string contents of this message. |
| `dest` | The recipient of the message. |
| `body_html` | The HTML string contents of this message. |
| `was_comment` | Whether this message was a comment. |
| `name` | The fullname of this message, representing the ID prefixed with its kind. (e.g. *t4_*) |
| `created` | The timestamp on which this message was created. |
| `created_utc` | The parsed UTC `datetime` on which this message was created. |
| `context` | `str` |
| `distinguished` | The type of distinguishment on this message object. |

**async author**() → *apraw.models.reddit.redditor.Redditor*

Retrieve the item's author as a `Redditor`.

**Returns author** – The item's author.

**Return type** *Redditor*

**async comment**(*text: str*) → Union[*Comment*, *Message*]
Reply to the item.

**Returns reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

**async fetch**()
Fetch this item's information from a suitable API endpoint.

**Returns self** – The updated model.

**Return type** *aPRAWBase*

**property fullname**
Get the ID prepended with its kind.

**Returns fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

**async reply**(*text: str*) → Union[*Comment*, *Message*]
Reply to the item.

**Returns reply** – The newly created reply, either a `Comment` or `Message`.

**Return type** *Comment* or *Message*

**async subreddit**() → *Subreddit*
Retrieve the subreddit this item was made in as a `Subreddit`.

**Returns subreddit** – The subreddit this item was made in.

**Return type** *Subreddit*

### 2.4.5 Redditor

This section describes the usage and members of the Redditor model.

A Redditor can be instantiated as follows:

```
sub = await reddit.redditor("aprawbot")
```

**class** apraw.models.**Redditor**(*reddit: Reddit*, *data: Dict*)
The model representing Redditors.

**reddit: Reddit** The `Reddit` instance with which requests are made.

**data: Dict** The data obtained from the /about endpoint.

**kind: str** The item's kind / type.

**subreddit: Sureddit** An instance of `Subreddit` for the Redditor's profile subreddit.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `comment_karma` | The amount of comment karma the Redditor has obtained. |
| `created_utc` | The date on which the Redditor was created in UTC `datetime`. |
| `created` | The timestamp of when the Redditor was created. |
| `has_verified_email` | Whether the Redditor has a verified email address. |
| `icon_img` | A URL to the Redditor's icon image if applicable. |
| `id` | The Redditor's ID (without kind). |
| `is_employee` | Whether the Redditor is a Reddit employee. |
| `is_friend` | Whether the Redditor has been added as a friend. |
| `is_gold` | Whether the Redditor is a Reddit gold member. |
| `is_mod` | Whether the Redditor is a moderator in a subreddit. |
| `is_suspended` | Whether the Redditor has been suspended. |
| `link_karma` | The amount of link karma the Redditor has obtained. |
| `name` | The Redditor's username. |
| `pref_show_snoovatar` | Whether to show the Redditor's Snoovatar in place of their icon. |
| `verified` | Whether the Redditor is verified. |

**Warning:** Suspended Redditors only return `is_suspended` and `name`.

`comments`(**args*, ***kwargs*)

Returns an instance of `ListingGenerator` mapped to fetch the Redditor's comments.

**Note:** This listing can be streamed doing the following:

```
for comment in redditor.comments.stream():
    print(comment)
```

**Parameters** **`kwargs`** (`**Dict`) – `ListingGenerator` `kwargs`.

**Returns** **generator** – A `ListingGenerator` mapped to fetch the Redditor's comments.

**Return type** *ListingGenerator*

`async fetch()`

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated `Redditor`.

**Return type** *Redditor*

`async message`(*subject*, *text*, *from_sr=''*) → Dict

Message the Redditor.

**Parameters**

- **`subject`** (`str`) – The subject of the message.
- **`text`** (`str`) – The text contents of the message in markdown.
- **`from_sr`** (`str`) – The subreddit the message is being sent from if applicable.

**Returns** **resp** – The response data returned from the endpoint.

**Return type** Dict

**moderated_subreddits**(***kwargs*) → *Subreddit*

Yields the subreddits the Redditor moderates.

**Parameters** **kwargs** (***Dict*) – kwargs to be used as query parameters.

**Yields** **subreddit** (*Subreddit*) – A subreddit the user moderates.

**submissions**(**args*, ***kwargs*)

Returns an instance of `ListingGenerator` mapped to fetch the Redditor's submissions.

**Note:** This listing can be streamed doing the following:

```
for comment in redditor.submissions.stream():
    print(comment)
```

**Parameters** **kwargs** (***Dict*) – `ListingGenerator` kwargs.

**Returns** **generator** – A `ListingGenerator` mapped to fetch the Redditor's submissions.

**Return type** *ListingGenerator*

### 2.4.6 MoreComments

This section describes the usage and members of the `MoreComments` model.

`MoreComments` stores a list of IDs pointing to `Comment` and further `MoreComments`. These can be retrieved using the `comments()` method or by iterating over the instance asynchronously:

```
comments = await more_comments.comments()
# or using asynchronous list comprehension:
comments = [c async for c in more_comments]
```

**class** apraw.models.**MoreComments**(*reddit: Reddit*, *data: Dict[str, Any]*, *link_id: str*)

Represents the model for more comments in a thread.

**Typical Attributes**

This table describes attributes that typically belong to objects of this class. Attributes are dynamically provided by the `aPRAWBase` class and may vary depending on the status of the response and expected objects.

| Attribute | Description |
|---|---|
| `count` | The number of comment or more children items in this thread. |
| `name` | The fullname that references this more comments model. |
| `id` | The ID of this more comments model. |
| `parent_id` | The ID of this more comments' parent submission or comment. |
| `depth` | The depth this more comments model goes into. |
| `children` | A list of comment and more comment IDs available in this thread. |

**async comments**() → List[Union[*apraw.models.reddit.comment.Comment*, *apraw.models.reddit.more_comments.MoreComments*]]

Retrieve a list of all the `Comment` and further `MoreComments` in this thread.

**Returns** **comments** – A list of all the `Comment` and further `MoreComments` in this thread.

**Return type** List[Union[*Comment*, *MoreComments*]]

`async fetch()`
    Fetch all the comments in this MoreComments thread.

`property fullname`
    Get the ID prepended with its kind.

    **Returns fullname** – The item's ID prepended with its kind such as *t1_*.

    **Return type** str

`async link()` → *Submission*
    Retrieve the submission this item belongs to as a `Submission`.

    **Returns submission** – The item's parent submission.

    **Return type** *Submission*

`async parent()` → Union[*apraw.models.reddit.submission.Submission*, *apraw.models.reddit.comment.Comment*]
    Retrieve the parent submission or comment of this MoreComments object.

    **Returns parent** – The parent submission or comment of this MoreComments object.

    **Return type** *Submission* or *Comment*

`async submission()` → *Submission*
    Retrieve the submission this item belongs to as a `Submission`.

    **Returns submission** – The item's parent submission.

    **Return type** *Submission*

### 2.4.7 Listing

Listings represent arrays returned by the Reddit API. It knows the `Reddit` instance it's working for, and contains references to `Subreddit` and `Submission` if available which are injected to the dynamically parsed aPRAW models.

Raw listings can be fetched with the `get_listing()` method where the endpoint needs to be supplied, and returns a listing.

`class apraw.models.Listing`(*reddit: Reddit*, *data: Dict*, *kind_filter: List[str] = None*, *subreddit:* Subreddit = *None*, *link_id: str = ''*)
    A model representing Reddit listings.

    `__getitem__`(*index: int*) → *apraw.models.helpers.apraw_base.aPRAWBase*
        Return the item at position index in the list.

        **Parameters index** (*int*) – The item's index.

        **Returns item** – The searched item.

        **Return type** *aPRAWBase*

    `__iter__()` → Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]
        Permit Listing to operate as an iterator.

        **Returns self** – The iterator.

        **Return type** *Listing*

    `__len__()` → int
        Return the number of items in the Listing.

        **Returns len** – The number of items in the listing.

**Return type** int

`__next__()` → *apraw.models.helpers.apraw_base.aPRAWBase*

Permit Listing to operate as a generator.

**Returns** **item** – The next item in the listing.

**Return type** *aPRAWBase*

`property last: apraw.models.helpers.apraw_base.aPRAWBase`

Return the last item in the listing.

**Returns** **item** – The last item in the listing.

**Return type** *aPRAWBase*

## 2.5 Helpers

This section contains the documentation of implemented base and helper classes used by aPRAW models.

### 2.5.1 ListingGenerator

`ListingGenerator` is a utility class that fetches items from the listing endpoint, parses the response, and yields items as they are found. If the item kind cannot be identified, `aPRAWBase` is returned which automatically assigns itself all the data attributes found.

`class apraw.models.ListingGenerator`(*reddit: Reddit*, *endpoint: str*, *limit: int = 100*, *subreddit:* Subreddit *= None*, *kind_filter: List[str] = None*, *listing_class: Type[*apraw.models.reddit.listing.Listing*] = apraw.models.reddit.listing.Listing*, ***kwargs*)

The model to request listings from Reddit.

**reddit: Reddit** The `Reddit` instance with which requests are made.

**endpoint: str** The endpoint to make requests on.

**max_wait: int** The maximum amount of seconds to wait before re-requesting in streams.

**kind_filter:** Kinds to return if given, otherwise all are returned.

**subreddit: Subreddit** The subreddit to inject as a dependency into items if given.

**Note:** ListingGenerator will automatically make requests until none more are found or the limit has been reached.

### 2.5.2 aPRAWBase

`aPRAWBase` is the base class used by most Reddit models to self-assign data retrieved from respective endpoints. It is used by classes such as `Submission` and `Comment`.

`class apraw.models.aPRAWBase`(*reddit: Reddit*, *data: Dict[str, Any] = None*, *kind: str = ''*)

The base class for Reddit models.

The `aPRAWBase` class stores data retrieved by the endpoints and automatically assigns it as attributes. Specific information about the aforementioned attributes can be found in the respective implementations such as `Comment`.

**kind: str** The item's kind / type.

`async fetch()`

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *aPRAWBase*

`property fullname`

Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

## 2.5.3 ItemModeration

`ItemModeration` is a utility class to aid in moderation comments, submissions and modmail. Specific implementations such as `CommentModeration` exist as well, and the base class may be used by certain models.

`class apraw.models.ItemModeration`(*reddit: Reddit*, *item:* apraw.models.helpers.apraw_base.aPRAWBase)

A helper class to moderate comments, submissions and modmail.

`async approve()`

Approve the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async distinguish`(*how:* apraw.models.enums.distinguishment_option.DistinguishmentOption *= 'yes'*, *sticky: bool = False*)

Distinguish the Reddit item.

**Parameters**

- **how** (`DistinguishmentOption`) – The type of distinguishment to be added to the item.
- **sticky** (*`bool, optional`*) – Whether the item should be stickied.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`property fullname: str`

Retrieve the fullname of the item this helper performs requests for.

**Returns** **fullname** – The ID prepended with the kind of the item this helper belongs to.

**Return type** str

`async ignore_reports()`

Ignore reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async remove`(*spam: bool = False*, *mod_note: Optional[str] = ''*, *reason: Optional[Union[str,* apraw.models.subreddit.removal_reasons.SubredditRemovalReason*]] = None*)

Remove the Reddit item.

**Parameters**

- **spam** (*`bool`*) – When `True`, use the removal to help train the Subreddit's spam filter (default: `False`).

- **mod_note** (*Optional[str]*) – A message for the other moderators.
- **reason** (*str or SubredditRemovalReason*) – The removal reason ID or a subreddit removal reason to add.

**Returns** **resp** – The API response JSON or a tuple of dictionaries if a removal reason / mod note was added as well.

**Return type** Dict or Tuple

`async` **`undistinguish`**`()`

Undistinguish the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

`async` **`unignore_reports`**`()`

Unignore previously ignored reports on the Reddit item.

**Returns** **resp** – The API response JSON.

**Return type** Dict

## 2.5.4 streamable

`streamable` is a callable class that can be used as a decorator on functions returning an asynchronous iterator. It is applied on functions such as *`new()`* and *`submissions()`*.

Streamable functions can be called by adding `.stream()`, for example `reddit.subreddits.new.stream()`.

`class apraw.models.`**`streamable`**(*func: Optional[Union[Callable[[Any, int, Any], Union[Awaitable[Union[AsyncIterator[*apraw.models.helpers.apraw_base.aPRAWBase*], Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]]], AsyncIterator[*apraw.models.helpers.apraw_base.aPRAWBase*], Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]]], AsyncGenerator[*apraw.models.helpers.apraw_base.aPRAWBase*, None], Generator[*apraw.models.helpers.apraw_base.aPRAWBase*, None, None]]] = None*, *max_wait: int = 16*, *attribute_name: str = 'fullname'*)

A decorator to add the `stream()` extension to functions returning (async) iterables or (async) generators.

**Parameters**

- **func** (*SYNC_OR_ASYNC_ITERABLE*) – The function returning an (async) iterable or (async) generator to be decorated.
- **max_wait** (*int*) – The maximum amount of time to wait in between requests that don't return new data.
- **attribute_name** (*str*) – The attribute to use as a unique identifier for items returned by the decorated function.

**Returns** **proxy** – A proxy descriptor that returns *`Streamable`* once it's accessed to enable per-instance use for bound methods and regular functions.

**Return type** ProxyStreamable

**class** `apraw.models.Streamable`(*func: Union[Callable[[Any, int, Any], Union[Awaitable[Union[AsyncIterator[*apraw.models.helpers.apraw_base.aPRAWBase*], Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]]], AsyncIterator[*apraw.models.helpers.apraw_base.aPRAWBase*], Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]]], AsyncGenerator[*apraw.models.helpers.apraw_base.aPRAWBase*, None], Generator[*apraw.models.helpers.apraw_base.aPRAWBase*, None, None]]*, *max_wait: int = 16*, *attribute_name: str = 'fullname'*, *instance: Optional[Any] = None*)

A decorator to make functions returning a generator streamable.

**max_wait: int** The maximum amount of seconds to wait before repolling the function.

**attribute_name: str** The attribute name to use as a unique identifier for returned objects.

**`__call__`**(**args*, ***kwargs*)

Make streamable callable to return result of decorated function.

**`stream`**(*skip_existing: bool = False*, **args*, ***kwargs*)

Call the stream method on the decorated function.

**Parameters**

- **`skip_existing`** (*`bool`*) – Whether items found before the function call should be returned as well.
- **`kwargs`** (*`**Dict`*) – `kwargs` to be passed on to the function.

**Yields** **item** (*aPRAWBase*) – The item retrieved by the function in chronological order.

### 2.5.5 CommentForest

**class** `apraw.models.CommentForest`(*reddit: Reddit*, *data: Dict*, *link_id: str*, *subreddit: apraw.models.subreddit.subreddit.Subreddit = None*)

`CommentForest` is an iterable used by *`Comment`* and *`Submission`* containing the replies and comment threads. The items can be iterated over just like any other listing, which could contain either *`Comment`* or *`MoreComments`*.

**`__getitem__`**(*index: int*) → *apraw.models.helpers.apraw_base.aPRAWBase*

Return the item at position index in the list.

**Parameters** **`index`** (*`int`*) – The item's index.

**Returns** **item** – The searched item.

**Return type** *aPRAWBase*

**`__iter__`**() → Iterator[*apraw.models.helpers.apraw_base.aPRAWBase*]

Permit Listing to operate as an iterator.

**Returns** **self** – The iterator.

**Return type** *Listing*

**`__len__`**() → int

Return the number of items in the Listing.

**Returns** **len** – The number of items in the listing.

**Return type** int

**`__next__`**() → *apraw.models.helpers.apraw_base.aPRAWBase*

Permit Listing to operate as a generator.

**Returns** **item** – The next item in the listing.

**Return type** *aPRAWBase*

`async fetch()`

Fetch this item's information from a suitable API endpoint.

**Returns** **self** – The updated model.

**Return type** *aPRAWBase*

`property fullname`

Get the ID prepended with its kind.

**Returns** **fullname** – The item's ID prepended with its kind such as *t1_*.

**Return type** str

`property last: apraw.models.helpers.apraw_base.aPRAWBase`

Return the last item in the listing.

**Returns** **item** – The last item in the listing.

**Return type** *aPRAWBase*

`async replace_more()`

Replaces all the `MoreComments` instances with the comments they reference.

This method can be used to retrieve all the comments, and only comments within a forest. This task could take a while for larger threads, after which the comment forest can be iterated over and all the comments with their replies will be made available.

## 2.6 Enums

This section contains the documentation of implemented enums used by methods to ensure sanitized arguments.

### 2.6.1 DistinguishmentOption

`DistinguishmentOption` is the enum used by `distinguish()` to distinguish submissions and comments with specific flags, or remove the distinguishment.

`class apraw.models.DistinguishmentOption(`*value*`)`

An enum for the distinguishment types.

- YES | "yes"
- NO | "no"
- ADMIN | "admin"
- SPECIAL | "special"

CHAPTER

THREE

# INDEX

- genindex

# INDEX